МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 «Программное обеспечение информационных

технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Реализация базы данных отеля с использованием технологии шифрования и дешифрования данных»

Выполнил студент Миневич Кристина Викторовна

(Ф.И.О.)

Руководитель проекта ассист. Нистюк Ольга Александровна

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

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

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

Курсовой проект защищен с оценкой

Минск 2022

**Содержание**

[Введение 3](#_Toc122221316)

[1. Постановка задачи 4](#_Toc122221317)

[1.1. Обзор прототипов и анализ 4](#_Toc122221318)

[1.1.1. DoubleTree by Hilton Minsk 4](#_Toc122221319)

[1.1.2. Minsk Marriott 4](#_Toc122221320)

[1.2. Аналогичность и различие программных средств 5](#_Toc122221321)

[2. Разработка модели базы данных 6](#_Toc122221322)

[2.1 Создание таблиц 6](#_Toc122221323)

[2.2 Создание представлений 9](#_Toc122221324)

[3. Разработка необходимых объектов 12](#_Toc122221325)

[3.1. Проектирование базы данных 12](#_Toc122221326)

[3.2. Процедуры для решения поставленных задач 13](#_Toc122221327)

[3.2.1. CRUD-процедуры таблиц 13](#_Toc122221328)

[3.2.2. Процедуры регистрации и авторизации 15](#_Toc122221329)

[3.2.3. Процедура подтверждения брони администратором 17](#_Toc122221330)

[3.2.4. Предоставление и изменение информации о пользователе 18](#_Toc122221331)

[3.3. Пользователи 19](#_Toc122221332)

[4. Описание процедур экспорта и импорта 21](#_Toc122221333)

[5. Описание и реализация технологий 23](#_Toc122221334)

[5.1. Шифрование данных 23](#_Toc122221335)

[5.2. Хранение мультимедийных типов данных 25](#_Toc122221336)

[6. Тестирование производительности 27](#_Toc122221341)

[7. Руководство пользователя по работе с программным средством 28](#_Toc122221342)

[Заключение 31](#_Toc122221343)

[Список использованной литературы 32](#_Toc122221344)

[Приложение A 33](#_Toc122221345)

[Приложение Б 34](#_Toc122221346)

[Приложение В 38](#_Toc122221347)

[Приложение Г 41](#_Toc122221348)

## **Введение**

Любая организация нуждается в своевременном доступе к информации. Ценность информации в современном мире очень высока. Роль распорядителей информации в современном мире чаще всего выполняют базы данных. Практически любая современная организация нуждается в базе данных, удовлетворяющей те или иные потребности по хранению, управлению и администрированию данных.

Целью данного проекта было проектирование с последующей реализацией базы данных на тему «База данных отеля». Она должна быть составлена для работы внутри отеля.

За основу базы данных «Отель» была взята модель реляционных баз данных.

База данных — это организованная структура, предназначенная для хранения информации, систематизированная таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины. Реляционная база данных – это набор данных с предопределенными связями между ними, после сравнений современных моделей баз данных, данная модель была выбрана для реализации проекта, реляционная модель отвечает необходимым требованиям: простота организации связей и понятное хранение данных в таблицах. Каждая строка представляет отдельную запись или элемент данных в таблице, который содержит значения для каждого из столбцов.

Среди огромного количества СУБД (Oracle, MySQL, MSSQL, PostgreSQL, MongoDB и другие) я выбрала Oracle Database 12c, так как в процессе учёбы приобрела навыки работы с этой СУБД.

Цель данного курсового проекта – создание базы данных для хранения пользовательской информации, списков свободных и зарезервированных комнат, наличие выбора сервисов для приобретения услуг, ознакомление с технологией шифрования и дешифрования данных и её применение в базе данных.

Основными задачами курсовой работы являются:

* + - провести аналитический обзор прототипов;
    - спроектировать базу данных;
    - реализовать функциональность базы данных;
    - провести тестирование используемой технологии в базе данных;
    - написать руководство пользователя.

Для успешного проектирования базы данных необходимо знать, из каких этапов оно состоит. Этапы проектирования базы данных:

* определение границ исследуемой области – предметной области;
* системный анализ – определение объектов и связей между ними;
* построение логической схемы базы данных в соответствии с определенными правилами – моделью данных;
* реализация базы данных – описание её в терминах некоторой СУБД.

## **Постановка задачи**

Целью данной работы является проектирование базы данных для программного средства отеля с технологией шифрования и дешифрования данных СУБД. В качестве модели данных следует использовать реляционную модель. Проектирование необходимо произвести таким образом, чтобы конечные данные соответствовали общим требованиям к информации в базе данных.

Функционал должен позволять:

* регистрировать нового пользователя;
* просматривать тип номеров;
* бронировать номер;
* заселять клиентов в номер;
* просматривать доступные сервисы;
* предоставлять сервисы (функция работника);
* просматривать заказанные сервисы (функция работника);
* подтверждать бронь номера (функция администратора);
* просматривать забронированные номера (функция администратора);
* выселять клиентов из номера (функция администратора);

Для наглядности весь доступный функционал будет представлен на UML-схеме (Приложение А).

Для реализации функционала необходимо разработать ряд объектов базы данных. К таким объектам относятся:

* пользователи;
* таблицы;
* представления;
* индексы;
* хранимые процедуры;
* функции;
* курсоры.

## **Обзор прототипов и анализ**

Одним из ключевых моментов в разработке программного обеспечения является просмотр и изучения различных аналогов, и поиск в них недостатков и достоинств. В наши дни множество подобных программ можно найти на сайтах с тематикой по данному профилю. Перед тем как приступить к работе требовалось провести анализ и прочитать соответствующие статьи по данной теме. В ходе поиска данной литературы мною были найдены прототипы сайтов некоторых отелей.

* + 1. **DoubleTree by Hilton Minsk**

Рассмотрим в качестве примера сайт известного отеля «DoubleTree» в Минске. Бронь комнаты осуществляется в 3 этапа:

1. Выбор номера (рисунок 1.1.1);

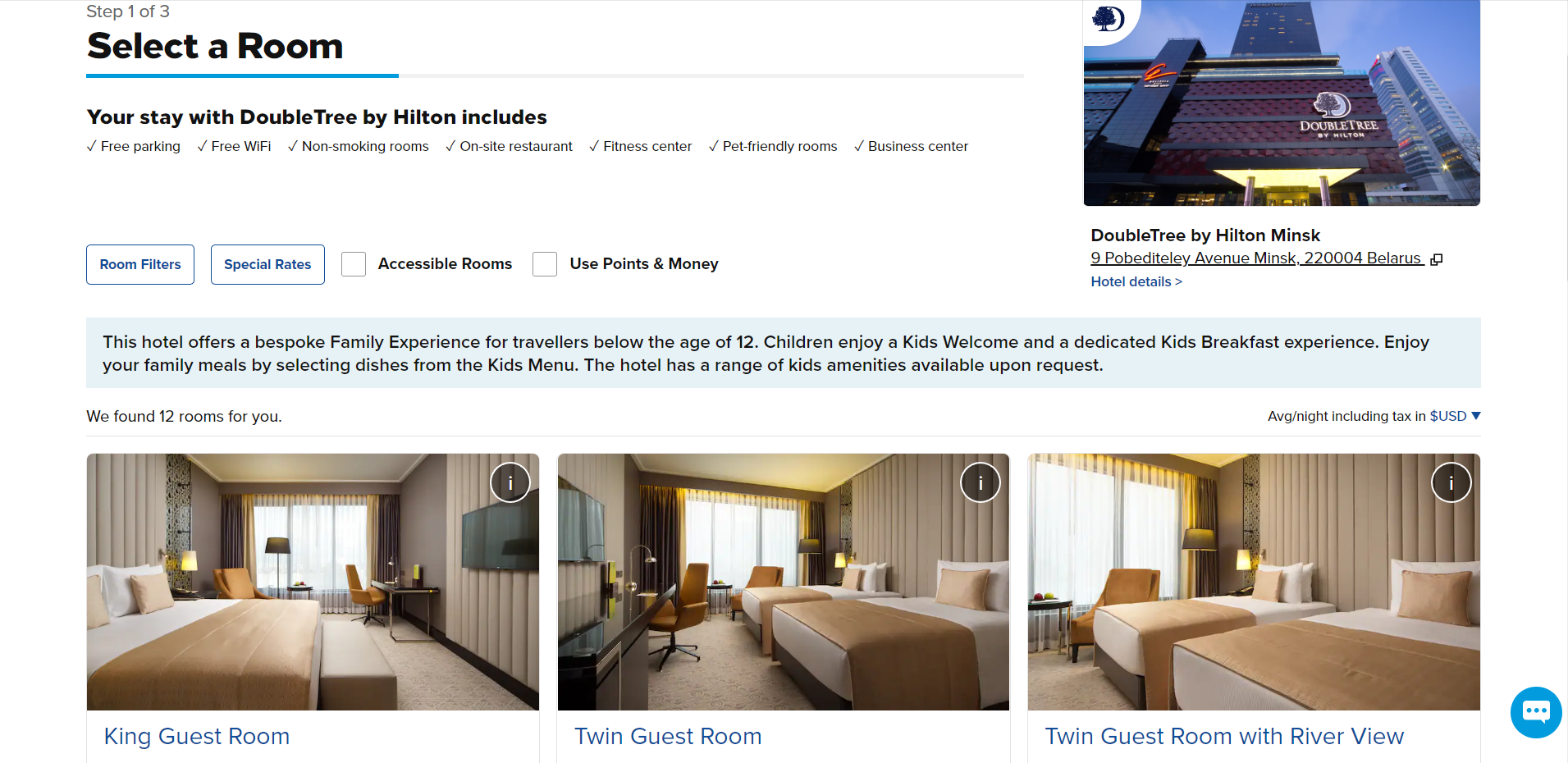


Рисунок 1.1.1 – Интерфейс выбора номера отеля DoubleTree

1. Выбор доступного тарифа (рисунок 1.1.2);

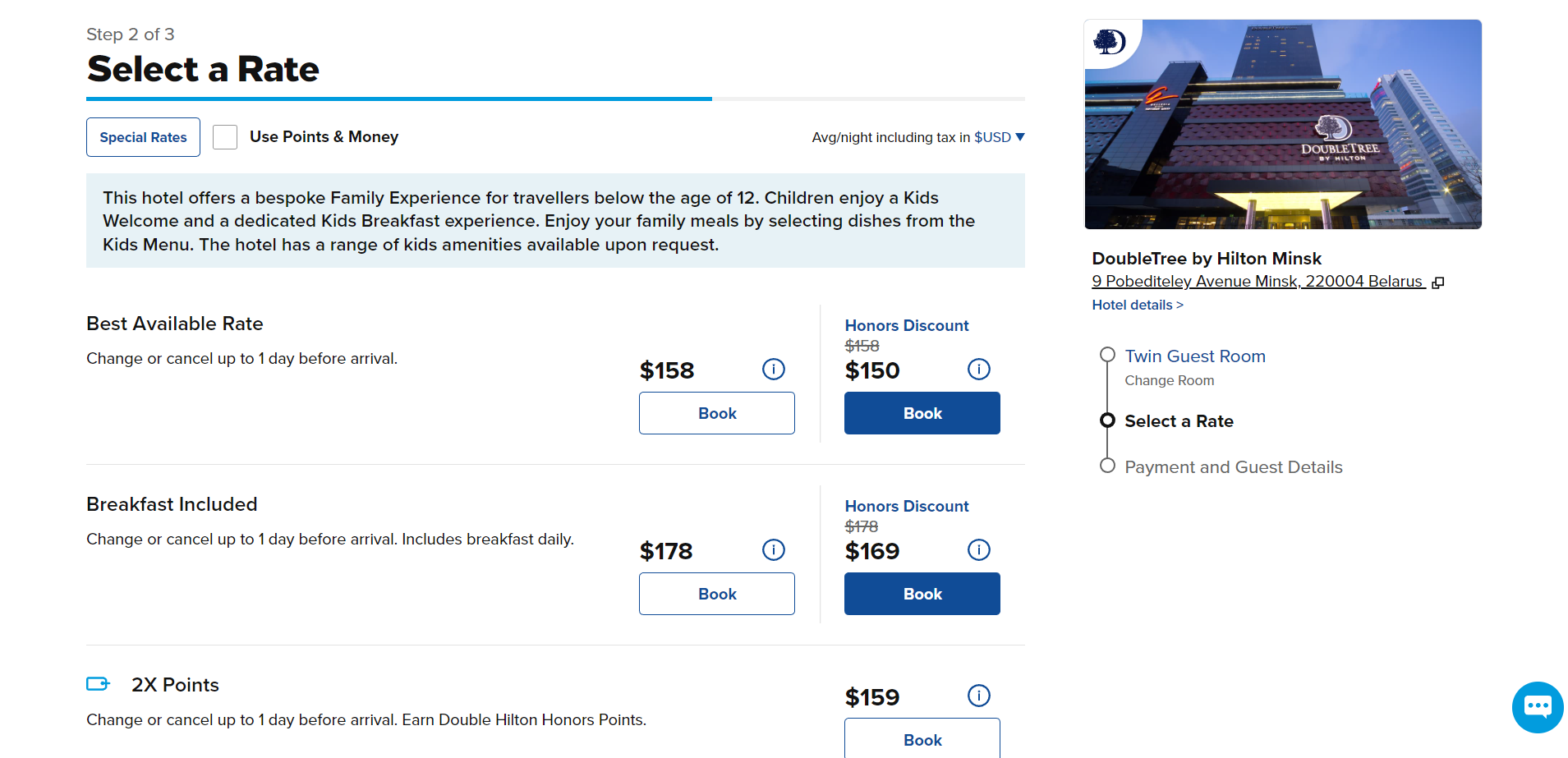


Рисунок 1.1.2 – Интерфейс выбора тарифа

3) Осуществление платежа и заполнение личной информации о клиенте (рисунок 1.1.3);

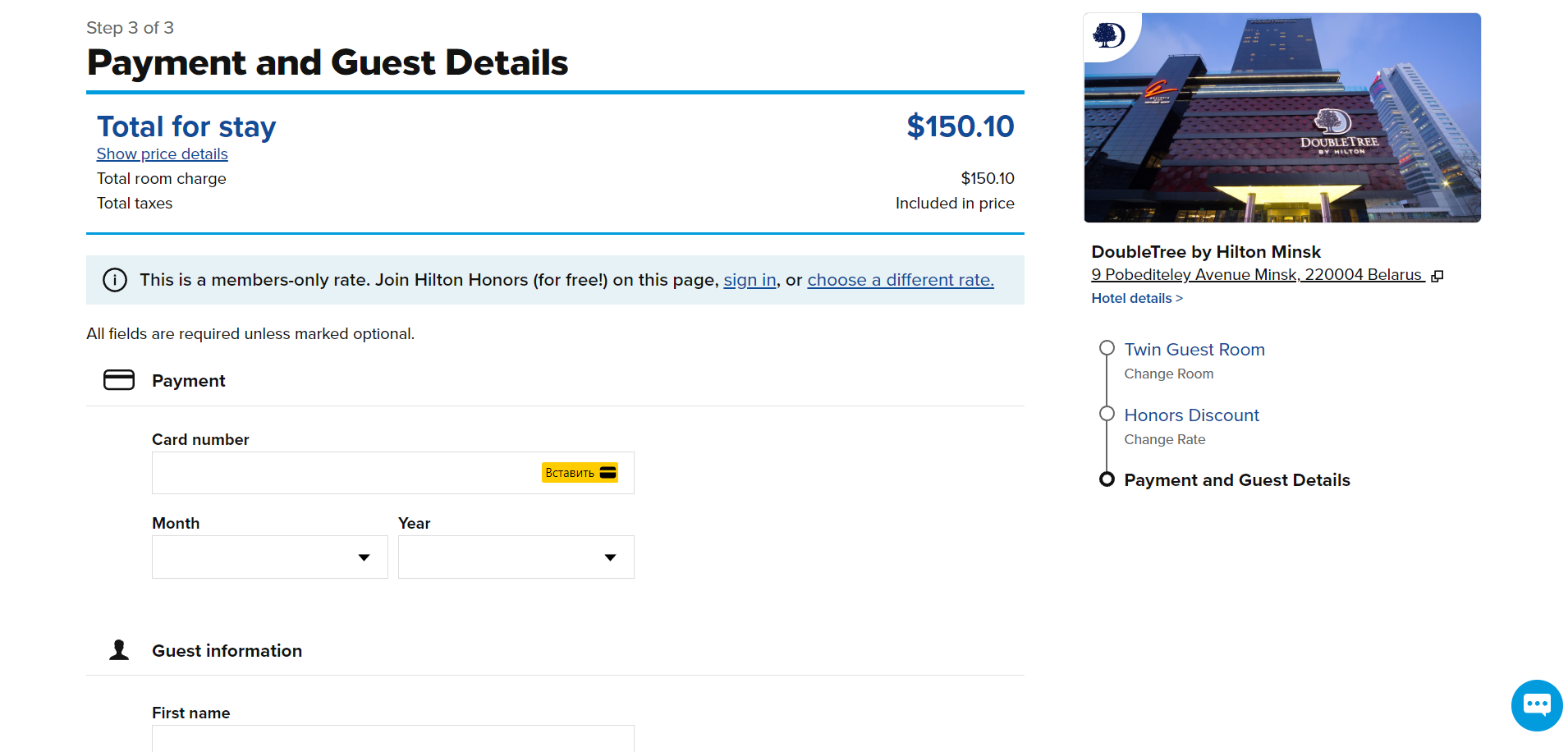


Рисунок 1.1.3 – Интерфейс заполнения формы о деталях брони

Из недостатков необходимо отметить небольшое количество сервисов, доступных в отеле: бесплатная парковка, Wi-Fi, возможность заселяться с питомцами, наличие ресторана внутри заведения.

* + 1. **Minsk Marriott**

Далее рассмотрим отель «Minsk Marriott». Для бронирования предлагается большое количество разнообразных номеров. Когда выбор номера сделан, появляется страница с описанием выбранной комнаты (рисунок 1.1.4).

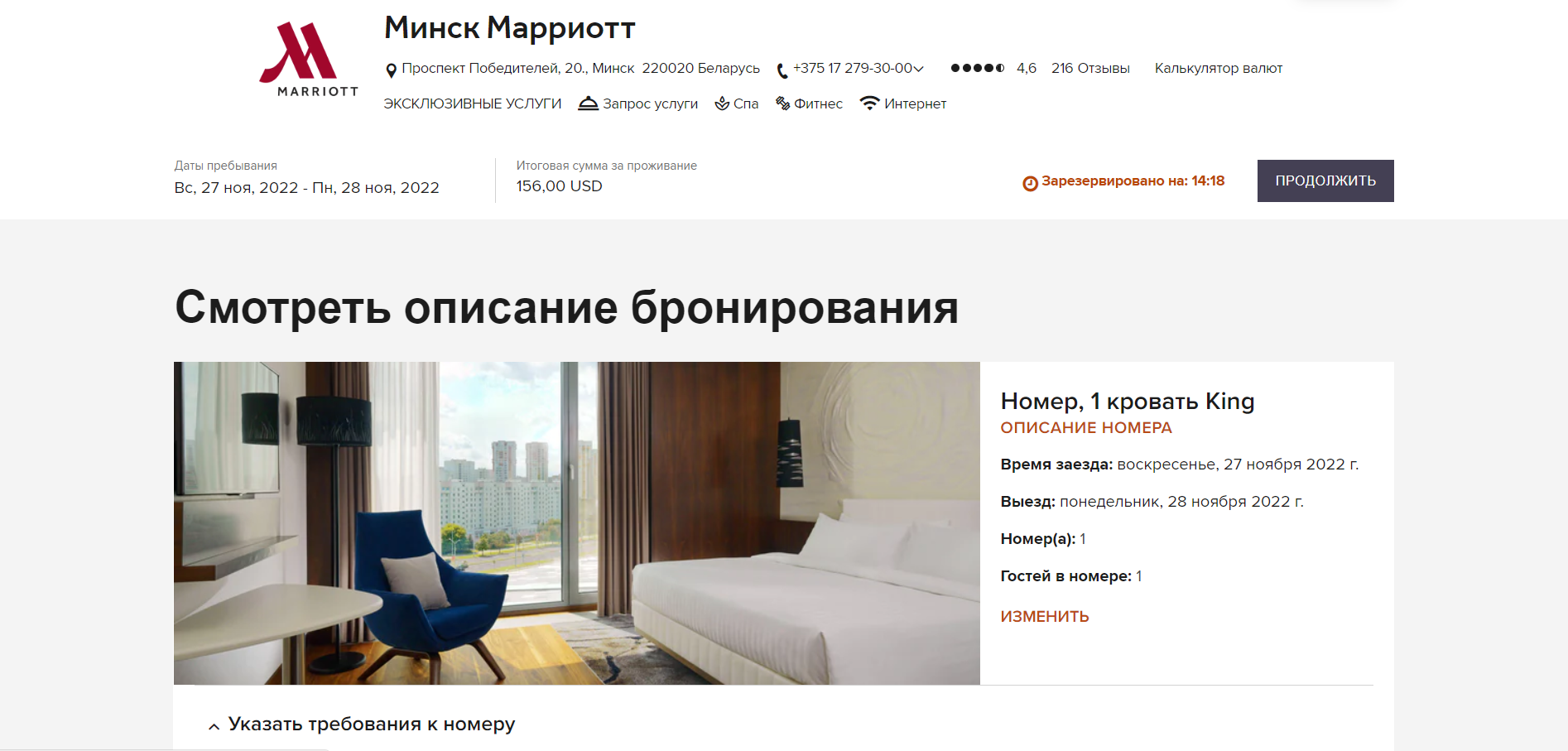


Рисунок 1.1.4 – Интерфейс страницы описания выбранного номера

Здесь же можно указать дополнительные услуги к бронированию (рисунок 1.1.5).

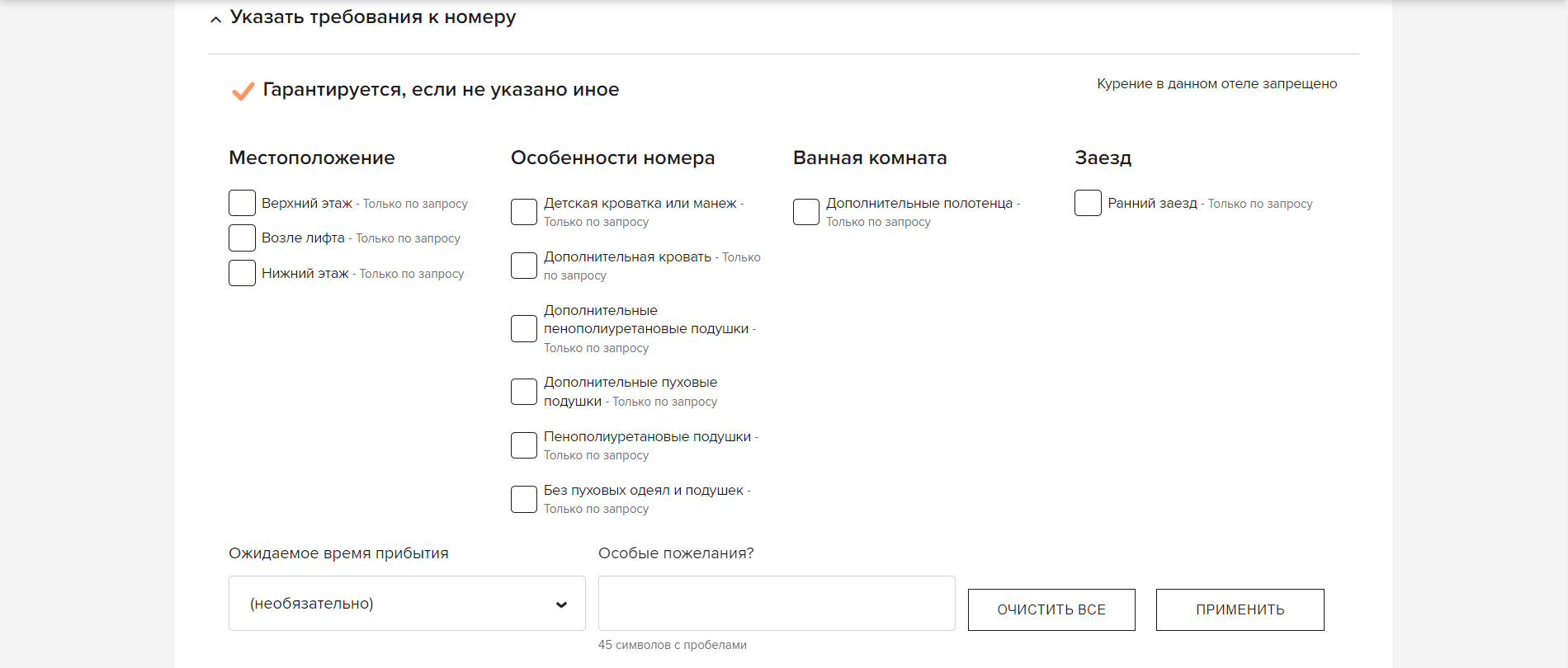


Рисунок 1.1.5 – Интерфейс формы дополнительных требований к брони

В конце нужно указать личные данные клиента и подтвердить бронь (рисунок 1.1.6).

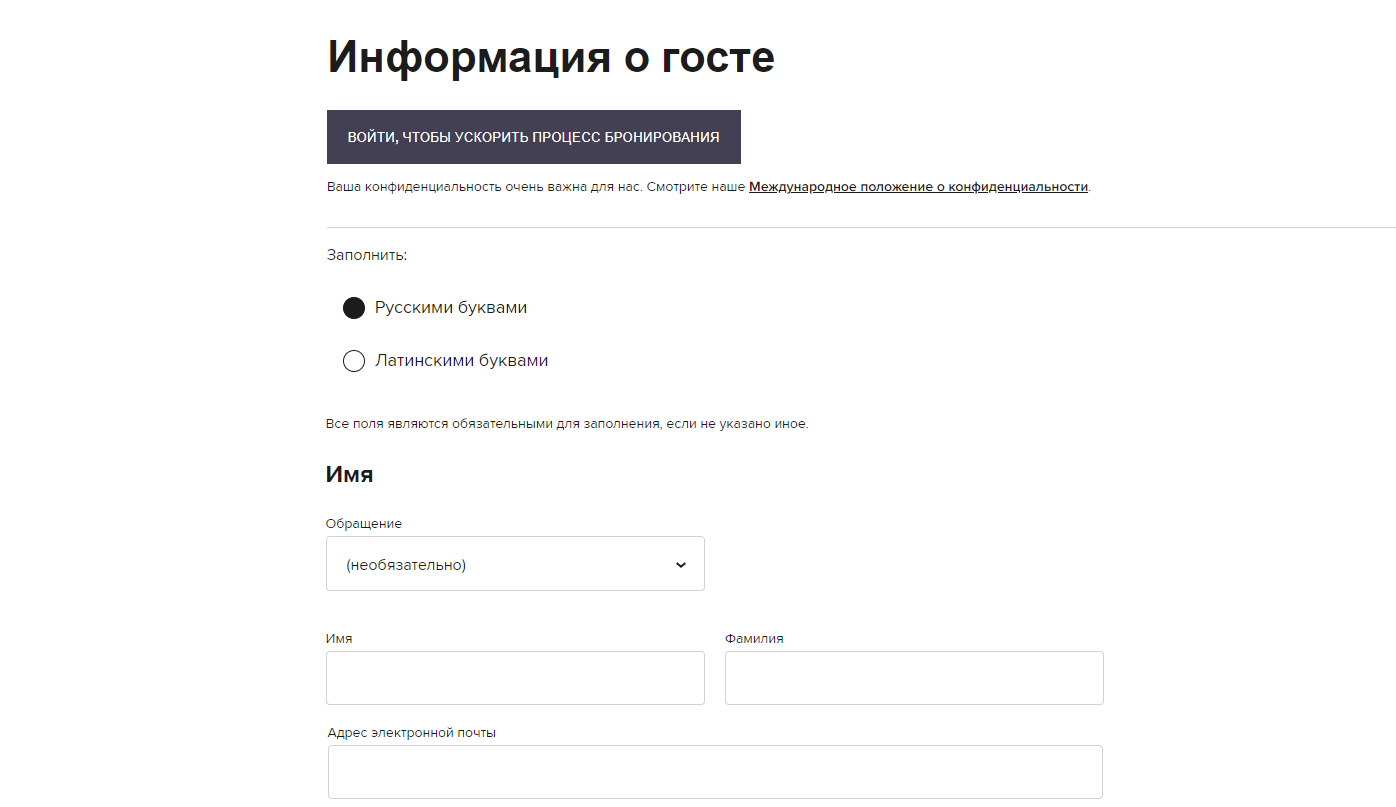


Рисунок 1.1.6 – Информация о госте

## **Аналогичность и различие программных средств**

Схожесть рассмотренных сайтов заключается в том, что все они имеют полный список комнат, а также осуществляют регистрацию клиента, заполняя личной информацией форму для подтверждения брони.

Главные отличия программ кроются в их функциональности. Так, например, сайт отеля «Minsk Marriott» предлагает выбор дополнительных сервисов после указания номера.

Анализируя ранее приведённые примеры, можно составить основные функциональные особенности для подобного типа приложений:

* регистрация и авторизация пользователя;
* указание нужных дат для брони, а также фильтр номеров (одноместные, двуместные и т.д.), а также предпочитаемый диапазон стоимости проживания;
* просмотр доступных комнат с учётом приведённых выше фильтров;
* указание при бронировании данных дополнительных проживающих в номере, сервисов и услуг, а также дата их бронирования.

Программа должна быть предназначена для различной аудитории пользователей. Это значит, что приложение должно быть простое и иметь доступный дизайн.

Все эти пункты и были учтены при выполнении данного курсового проекта.

## **Разработка модели базы данных**

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

* таблицы;
* представления;
* пользователи;
* хранимые процедуры;
* функции;
* курсоры.

## **2.1 Создание таблиц**

Первым этапом курсового проекта будет создание логически взаимосвязанных таблиц. Чтобы составить визуальную взаимосвязанную структуру базы данных, нам необходимо продумать, какая информация будет храниться в этих таблицах, после этого создать связи с помощью первичных и внешних ключей.

Для реализации работы базы данных было создано 14 таблиц.

Диаграмма базы со структурой связей представлена в приложении А.

Логически можно вывести 8 основных таблиц: PERSON\_TABLE, ROOM\_TABLE, BOOKING\_TABLE, SERVICE\_TABLE, RENT\_TABLE, INVENTORY\_TABLE, PHOTO\_TABLE, ALBUM\_TABLE.

Таблица **PERSON\_TABLE** представляет собой данные о пользователе, состоит из столбцов:

* person\_id – идентификатор пользователя, тип number(10), первичный ключ;
* person\_role\_id – идентификатор роли, тип number(10), внешний ключ;
* person\_email – адрес электронной почты пользователя, тип nvarchar2(50);
* person\_password – пароль пользователя, тип nvarchar2(50);
* person\_first\_name – имя пользователя, тип nvarchar2(50);
* person\_last\_name – фамилия пользователя, тип nvarchar2(50);
* person\_father\_name – отчество пользователя, тип nvarchar2(50).

Таблица **ROOM\_TABLE** представляет собой информацию о номерах, состоит из столбцов:

* room\_id – идентификатор номера, тип number(10), первичный ключ;
* room\_room\_type\_id – идентификатор типа номера, тип number(10), внешний ключ;
* room\_album\_id – идентификатор альбома номера, тип number(10), внешний ключ;
* room\_number – номер комнаты, тип nvarchar2(50);
* room\_description – описание номера, тип nvarchar2(200);

Таблица **BOOKING\_TABLE** предназначена для хранения информации о бронировании номера, состоит из следующих столбцов:

* booking\_id – идентификатор брони, тип number(10), первичный ключ;
* booking\_room\_id – идентификатор забронированного номера, тип number(10), внешний ключ;
* booking\_start\_date – дата начала бронирования, тип date;
* booking\_end\_date – дата окончания бронирования, тип date;
* booking\_state – статус бронирования, тип number(1) (0 – забронировано клиентом, 1 – одобрено администратором).

Таблица **SERVICE\_TABLE** представляет собой информацию о сервисах, состоит из столбцов:

* service\_id – идентификатор сервиса, тип number(10), первичный ключ;
* service\_service\_type\_id – идентификатор типа сервиса, тип number(10), внешний ключ;
* service\_person\_id – идентификатор человека, забронировавшего сервис, тип number(10), внешний ключ.

Таблица **RENT\_TABLE** представляет собой информацию об арендованном инвентаре, состоит из столбцов:

* rent\_id – идентификатор аренды, тип number(10), первичный ключ;
* rent\_inventory\_id – идентификатор инвентаря, тип number(10), внешний ключ;
* rent\_booking\_id – идентификатор брони номера, тип number(10), внешний ключ;
* rent\_start\_date – дата начала арендования, тип date;
* rent\_end\_date – дата окончания арендования, тип date.

Таблица **INVENTORY\_TABLE** представляет собой инвентарь, состоит из столбцов:

* inventory\_id – идентификатор инвентаря, тип number(10), первичный ключ;
* inventory\_inventory \_type\_id – идентификатор типа инвентаря, тип number(10), внешний ключ;
* inventory\_album\_id – идентификатор альбома инвентаря, тип number(10), внешний ключ;
* inventory\_description – описание инвентаря, тип nvarchar2(200);
* inventory\_daily\_price – стоимость ежедневного пользования инвентарём, тип float(10).

Таблица **PHOTO\_TABLE** содержит информацию о фотографиях, состоит из столбцов:

* photo\_id – идентификатор фото, тип number(10), первичный ключ;
* photo\_album\_id – идентификатор альбома, тип number(10), внешний ключ;
* photo\_source – название фото, тип blob.

Таблица **ALBUM\_TABLE** содержит информацию об альбомах, состоит из столбцов:

* album\_id – идентификатор альбома, тип number(10), первичный ключ;
* album\_name – название альбома, тип nvarchar2(50);
* album\_object – объект, которому принадлежит альбом, тип number(10).

Также имеется ещё 6 таблиц для поддержания всей остальной логической структуры базы данных отеля: ROLE\_TABLE, ROOM\_TYPE\_TABLE, SERVICE\_TYPE\_TABLE, RESIDENT\_TABLE, INVENTORY\_TYPE\_TABLE, SUBSCRIPTION\_TABLE.

Таблица **ROLE\_TABLE** представляет собой роль, состоит из столбцов:

* role\_id – идентификатор роли, тип number(10), первичный ключ;
* role\_name – название роли, тип nvarchar2(50), внешний ключ.

Таблица **ROOM\_TYPE\_TABLE** представляет собой тип комнаты, состоит из столбцов:

* room\_type\_id – идентификатор типа комнаты, тип number(10), первичный ключ;
* room\_type\_name – название типа комнаты, тип nvarchar2(50);
* room\_type\_capacity – вместимость комнаты (1 – одноместная, 2 – двуместная, 3 – трёхместная и т.д.), тип number(10);
* room\_type\_daily\_price – ежедневная стоимость данного типа комнаты, тип float(10).

Таблица **SERVICE\_TYPE\_TABLE** представляет собой тип сервиса, состоит из столбцов:

* service\_type\_id – идентификатор типа сервиса, тип number(10), первичный ключ;
* service\_type\_name – название типа сервиса, тип nvarchar2(50);
* service\_type\_daily\_price – ежедневная стоимость данного типа сервиса, тип float(10).

Таблица **RESIDENT\_TABLE** представляет собой информацию о жителях комнаты, состоит из столбцов:

* resident\_id – идентификатор жителей комнаты, тип number(10), первичный ключ;
* resident\_person\_id – идентификатор одного жителя комнаты, тип number(10), внешний ключ;
* resident\_booking\_id – идентификатор брони жителей, внешний ключ.

Таблица **INVENTORY\_TYPE\_TABLE** содержит информацию о типах имеющихся инвентарей, состоит из столбцов:

* inventory\_id – идентификатор инвентаря, тип number(10), первичный ключ;
* inventory\_type\_name – название типа инвентаря, тип nvarchar2(50).

Таблица **SUBSCRIPTION\_TABLE** содержит информацию о подписках на определённый сервис, состоит из столбцов:

* subscription\_id – идентификатор подписки, тип number(10), первичный ключ;
* subscription\_service\_id – идентификатор сервиса, тип number(10), внешний ключ;
* subscription\_booking\_id – идентификатор брони номера, тип number(10), внешний ключ;
* subscription\_start\_date – дата начала подписки, тип date;
* subscription\_end\_date – дата окончания подписки, тип date.

## **2.2 Создание представлений**

Для упрощения пользовательских запросов, предотвращения ошибок, улучшения производительности и преобразования данных в более понятный вид были созданы представления к каждой таблице базы данных. Рассмотрим примеры нескольких из них.

Представление person\_view соединяет между собой таблицы person\_table, role\_table, album\_table и позволяет выбрать данные по столбцам из этих таблиц, а именно: идентификатор пользователя (person\_table.person\_id), email пользователя (person\_table.person\_email), пароль пользователя (person\_table.person\_password), имя пользователя (person\_table.person\_first\_name), фамилия пользователя (person\_table.person\_last\_name), отчество пользователя (person\_table.person\_father\_name), идентификатор роли (role\_table.role\_id), имя роли (role\_table.role\_name), идентификатор альбома (album\_table.album\_id), название альбома (album\_table.album\_name), объект, которому принадлежит альбом (album\_table.album\_object).

**CREATE VIEW** person\_view   
**AS SELECT** person\_table.**person\_id**, person\_table.**person\_email**, person\_table.**person\_password**, person\_table.**person\_first\_name**, person\_table.**person\_last\_name**, person\_table.**person\_father\_name**, *--person\_table fields*role\_table.**role\_id**, role\_table.**role\_name**, *--role\_table fields*album\_table.**album\_id**, album\_table.**album\_name**, album\_table.**album\_object** *--album\_table fields***FROM** person\_table **INNER JOIN** role\_table **ON** person\_table.**person\_role\_id** = role\_table.**role\_id  
LEFT OUTER JOIN** album\_table **ON** person\_table.**person\_album\_id** = album\_table.**album\_id**;

Листинг 2.2.1 – Создание представления person\_view

Представление room\_view соединяет между собой таблицы room\_table, room\_type\_table, album\_table и позволяет выбрать данные по столбцам из этих таблиц: идентификатор комнаты (room\_table.room\_id), номер комнаты (room\_table.room\_number), описание комнаты (room\_table.room\_description),

идентификатор типа комнаты (room\_type\_table.room\_type\_id), тип комнаты (room\_type\_table.room\_type\_name), вместимость комнаты (room\_type\_table.room\_capacity), ежедневная плата за проживание в комнате определённого типа (room\_type\_table.room\_type\_daily\_price), идентификатор альбома (album\_view.album\_id), название альбома (album\_view.album\_name), а также объект, которому принадлежит альбом (album\_table.album\_object).

**CREATE VIEW** room\_view   
**AS SELECT** room\_table.**room\_id**, room\_table.**room\_number**, room\_table.**room\_description**, *--room\_table fields*room\_type\_table.**room\_type\_id**, room\_type\_table.**room\_type\_name**, room\_type\_table.**room\_type\_capacity**, room\_type\_table.**room\_type\_daily\_price**, *--room\_type\_table fields*album\_table.**album\_id**, album\_table.**album\_name**, album\_table.**album\_object**

*--album\_table fields***FROM** room\_table **INNER JOIN** room\_type\_table **ON** room\_table.**room\_room\_type\_id** = room\_type\_table.**room\_type\_id  
LEFT OUTER JOIN** album\_table **ON** room\_table.**room\_album\_id** = album\_table.**album\_id**;

Листинг 2.2.2 – Создание представления room\_view

Представление inventory\_view соединяет между собой таблицы и представления, такие, как inventory\_type\_table, inventory\_table и album\_view. Это представление позволяет выбрать данные по столбцам из этих таблиц, а именно: идентификатор типа инвентаря (inventory\_type\_table.inventory\_type\_id), название типа инвентаря (inventory\_type\_table.inventory\_type\_name), идентификатор инвентаря (inventory\_table.inventory\_id), описание инвентаря (inventory\_table.inventory\_description), ежедневная плата за инвентарь (inventory\_table.inventory\_daily\_price), идентификатор альбома (album\_view.album\_id), название альбома (album\_view.album\_name).

**CREATE VIEW** inventory\_view   
**AS SELECT** inventory\_type\_table.**inventory\_type\_id**, inventory\_type\_table.**inventory\_type\_name**, *--inventory\_type\_table fields*inventory\_table.**inventory\_id**, inventory\_table.**inventory\_description**, inventory\_table.**inventory\_daily\_price**, *--inventory\_table fields*album\_table.**album\_id**, album\_table.**album\_name**, album\_table.**album\_object** *--album\_table fields***FROM** inventory\_type\_table **INNER JOIN** inventory\_table **ON** inventory\_type\_table.**inventory\_type\_id** = inventory\_table.**inventory\_inventory\_type\_id  
LEFT OUTER JOIN** album\_table **ON** inventory\_table.**inventory\_album\_id** = album\_table.**album\_id**;

Листинг 2.2.3 – Создание представления inventory\_view

Представление service\_view соединяет между собой таблицы и представления service\_type\_table, service\_table и person\_view. Данные выбираются со следующих столбцов: идентификатор типа сервиса (service\_type\_table.service\_type\_id), название типа сервиса (service\_type\_table.service\_type\_name), ежедневная плата за определённый вид сервиса (service\_type\_table.service\_type\_daily\_price), идентификатор сервиса (service\_table.service\_id), идентификатор пользователя (person\_view.person\_id), email пользователя (person\_view.person\_email), пароль пользователя (person\_view.person\_password), имя пользователя (person\_view.person\_first\_name), фамилия пользователя (person\_view.person\_last\_name), отчество пользователя (person\_view.person\_father\_name), идентификатор роли пользователя (person\_view.role\_id), название роли (person\_view.role\_name).

**CREATE VIEW** service\_view   
**AS SELECT** service\_type\_table.**service\_type\_id**, service\_type\_table.**service\_type\_name**, service\_type\_table.**service\_type\_daily\_price**, *--service\_type\_table fields*service\_table.**service\_id**, *--service\_table fields*person\_view.**person\_id**, person\_view.**person\_email**, person\_view.**person\_password**, person\_view.**person\_first\_name**, person\_view.**person\_last\_name**, person\_view.**person\_father\_name**, *--person\_table fields*person\_view.**role\_id**, person\_view.**role\_name**, *--role\_table fields*person\_view.**album\_id**, person\_view.**album\_name**, person\_view.**album\_object** *--album\_table fields***FROM** service\_type\_table **INNER JOIN** service\_table **ON** service\_type\_table.**service\_type\_id** = service\_table.**service\_service\_type\_id  
LEFT OUTER JOIN** person\_view **ON** service\_table.**service\_person\_id** = person\_view.**person\_id**;

Листинг 2.2.4 – Создание представления service\_view

## **Разработка необходимых объектов**

## **Проектирование базы данных**

Для реализации поставленной задачи была создана база данных Hotel. Для её создания использовалась система управления реляционными базами данных Oracle Database 12c.

Прежде всего, необходимо было спроектировать корректную базу данных для работы. Моя база данных содержит следующие таблицы (рисунок 3.1):

* роли (role\_table);
* информация о жильцах (person\_table);
* комнаты (room\_table);
* тип комнат (room\_type\_table);
* сервисы (service\_table);
* тип сервиса (service\_type\_table);
* инвентарь (inventory\_table);
* тип инвентаря (inventory\_type\_table);
* жильцы (resident\_table);
* забронированные комнаты (booking\_table);
* аренда инвентаря (rent\_room);
* подписки на сервис (subscription\_table);
* фото из альбомов (photo\_table);
* альбомы (album\_table).



Рисунок 3.1 – Диаграмма базы данных

Листинг создания таблиц базы данных представлен в приложении Б.

## **Процедуры для решения поставленных задач**

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

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.

При разработке курсового проекта было создано 50 процедур для следующих целей:

1. Выборка данных из таблиц по различным полям;
2. Добавление пользователя/проверка при авторизации;
3. Удаление данных из таблиц;
4. Добавление данных в таблицы;
5. Изменение данных в таблицах;
6. Экспорт и импорт таблиц в формат xml;
7. Заполнение таблицы на 100 000 строк.

Весь перечень созданных процедур будет представлен в Приложении В.

Все процедуры можно разделить на 3 категории для выполнения следующих задач:

1. CRUD-процедуры для каждой таблицы (create, update, delete);

2. Процедуры, реализующие выборку данных по запросу;

3. Процедуры выполняющие операции, которые нельзя отнести ни в одну из категорий, вынесены в специальные процедуры.

Далее мы более подробно остановимся на каждой из категорий.

## **CRUD-процедуры таблиц**

Все CRUD-процедуры для каждой операции для всех таблиц построены по одному шаблону, за исключением, для операций CREATE они принимают параметры для каждого поля таблицы или те, которые должны быть обновлены для UPDATE. Уникальный идентификатор в качестве параметра передаётся для операций UPDATE и DELETE.

Чтобы избежать повторений и из-за шаблонности CRUD-процедур, все примеры будут рассмотрены на примере процедур для одной таблицы – таблица ROOM\_TYPE\_TABLE.

Все CREATE-процедуры принимают в качестве параметров все поля, кроме идентификатора, описывающие таблицу, пример скрипта представлен в листинге 3.1.

**CREATE OR REPLACE PROCEDURE** *create\_room\_type* (   
 p\_room\_type\_name **IN** room\_type\_table.**room\_type\_name**%**TYPE**,  
 p\_room\_type\_daily\_price **IN** room\_type\_table.**room\_type\_daily\_price**%**TYPE**

Продолжениe листинга 3.1

)  
**IS** cnt **NUMBER**;  
**BEGIN  
 SELECT** *COUNT*(*\**) **INTO** cnt **FROM** room\_type\_table **WHERE** *UPPER*(**room\_type\_name**) = *UPPER*(p\_room\_type\_name);  
 **IF** (cnt = 0) **THEN  
 IF** (p\_room\_type\_daily\_price > 0) **THEN  
 INSERT INTO** room\_type\_table(**room\_type\_name**, **room\_type\_daily\_price**) **VALUES**(p\_room\_type\_name, p\_room\_type\_daily\_price);  
 **COMMIT**;  
 **ELSE** *RAISE\_APPLICATION\_ERROR*(-20006, **'The price should be positive number'**);  
 **END IF**;  
 **ELSE** *RAISE\_APPLICATION\_ERROR*(-20005, **'This room type already exists'**);  
 **END IF**;  
**END** *create\_room\_type*;

Листинг 3.1 – СREATE-процедура на примере процедуры create\_room\_type

Далее будет рассмотрен пример UPDATE операции. Для обновления информации в нужной нам таблице, мы должны знать идентификатор изменяемой нами строки, который мы передаём в параметры нашей процедуры вместе с другими параметрами. Каждый параметр соответствует полю таблицы. Пример скрипта представлен в листинге 3.2.

**CREATE OR REPLACE PROCEDURE** *update\_room\_type* (  
 p\_room\_type\_id **IN** room\_type\_table.**room\_type\_id**%**TYPE**,  
 p\_room\_type\_name **IN** room\_type\_table.**room\_type\_name**%**TYPE**,  
 p\_room\_type\_daily\_price **IN** room\_type\_table.**room\_type\_daily\_price**%**TYPE** )  
**IS** cnt **NUMBER**;  
**BEGIN  
 SELECT** *COUNT*(*\**) **INTO** cnt **FROM** room\_type\_table **WHERE room\_type\_id** = p\_room\_type\_id;  
 **IF** (cnt != 0) **THEN  
 SELECT** *COUNT*(*\**) **INTO** cnt **FROM** room\_type\_table **WHERE** *UPPER*(**room\_type\_name**) = *UPPER*(p\_room\_type\_name) **AND room\_type\_id** != p\_room\_type\_id;  
 **IF** (cnt = 0) **THEN  
 IF** (p\_room\_type\_daily\_price > 0) **THEN  
 UPDATE** room\_type\_table **SET room\_type\_name** = p\_room\_type\_name, **room\_type\_daily\_price** = p\_room\_type\_daily\_price **WHERE**

Продолжениe листинга 3.2

**room\_type\_id** = p\_room\_type\_id;  
 **COMMIT**;  
 **ELSE** *RAISE\_APPLICATION\_ERROR*(-20006, **'The price should be positive number'**);  
 **END IF**;   
 **ELSE** *RAISE\_APPLICATION\_ERROR*(-20005, **'This room type already exists'**);  
 **END IF**;  
 **ELSE** *RAISE\_APPLICATION\_ERROR*(-20007, **'Room type is not found'**);  
 **END IF**;  
**END** *update\_room\_type*;

Листинг 3.2 – UPDATE-процедура на примере процедуры update\_room\_type

Все DELETE-процедуры принимают один параметр – уникальный идентификатор записи таблицы, после чего осуществляют поиск по ней, найдя запись для удаления – удаляют её. Пример скрипта приведён в листинге 3.3.

**CREATE OR REPLACE PROCEDURE** *delete\_room\_type* (  
 p\_room\_type\_id **IN** room\_type\_table.**room\_type\_id**%**TYPE** )  
**IS** cnt **NUMBER**;  
**BEGIN  
 SELECT** *COUNT*(*\**) **INTO** cnt **FROM** room\_type\_table **WHERE room\_type\_id** = p\_room\_type\_id;  
 **IF** (cnt != 0) **THEN  
 DELETE FROM** room\_type\_table **WHERE room\_type\_id** = p\_room\_type\_id;  
 **COMMIT**;  
 **ELSE** *RAISE\_APPLICATION\_ERROR*(-20007, **'Room type is not found'**);  
 **END IF**;  
**END** *delete\_room\_type*;

Листинг 3.3 – DELETE-процедура на примере процедуры delete\_room\_type.

## **Процедуры регистрации и авторизации**

При регистрации нового пользователя с помощью процедуры register\_person, ему необходимо заполнить следующие поля: email, password, first\_name, last\_name, father\_name. Для увеличения безопасности учетной записи пользователя его пароль шифруется с помощью алгоритма AES128. Также проверяется корректность ввода адреса электронной почты на повторение. Данная процедура представлена ниже листинге 3.4:

**CREATE OR REPLACE PROCEDURE** *register\_person* (   
 p\_person\_email **IN** person\_table.**person\_email**%**TYPE**,  
 p\_person\_password **IN** person\_table.**person\_password**%**TYPE**,  
 p\_person\_first\_name **IN** person\_table.**person\_first\_name**%**TYPE**,  
 p\_person\_last\_name **IN** person\_table.**person\_last\_name**%**TYPE**,  
 p\_person\_father\_name **IN** person\_table.**person\_father\_name**%**TYPE** )  
**IS** cnt **NUMBER**;  
**BEGIN  
 SELECT** *COUNT*(*\**) **INTO** cnt **FROM** person\_table **WHERE** *UPPER*(**person\_email**) = *UPPER*(p\_person\_email);  
 **IF** (cnt = 0) **THEN  
 INSERT INTO** person\_table(**person\_email**, **person\_password**, **person\_first\_name**, **person\_last\_name**, **person\_father\_name**)   
 **VALUES**(p\_person\_email, *encryption\_password*(p\_person\_password), p\_person\_first\_name, p\_person\_last\_name, p\_person\_father\_name);  
 **COMMIT**;  
 **ELSE** *RAISE\_APPLICATION\_ERROR*(-20001, **'This person already exists'**);  
 **END IF**;  
**END** *register\_person*;

Листинг 3.4 – Описание процедуры register\_person

В случае успешной регистрации процедура создаст нового пользователя с ролью Guest. Для дальнейшей работы необходимо авторизоваться с помощью процедуры log\_in\_person, представленной в листинке 3.5. В качестве входных параметров используются p\_person\_email и p\_person\_password.

**CREATE OR REPLACE PROCEDURE** *log\_in\_person* (  
 p\_person\_email **IN** person\_table.**person\_email**%**TYPE**,  
 p\_person\_password **IN** person\_view.**person\_password**%**TYPE**,  
 o\_person\_id **OUT** person\_table.**person\_id**%**TYPE**,  
 o\_person\_email **OUT** person\_table.**person\_email**%**TYPE**,  
 o\_person\_password **OUT** person\_view.**person\_password**%**TYPE**,  
 o\_person\_first\_name **OUT** person\_view.**person\_first\_name**%**TYPE**,  
 o\_person\_last\_name **OUT** person\_view.**person\_last\_name**%**TYPE**,  
 o\_person\_father\_name **OUT** person\_view.**person\_father\_name**%**TYPE**,  
 o\_role\_name **OUT** person\_view.**role\_name**%**TYPE** )  
**IS  
 CURSOR** person\_cursor **IS SELECT person\_id**, **person\_email**, p\_person\_password, **person\_first\_name**, **person\_last\_name**, **person\_father\_name**,

Продолжениe листинга 3.5

**role\_name FROM** person\_view **WHERE** *UPPER*(**person\_email**) = *UPPER*(p\_person\_email) **AND person\_password** = *encryption\_password*(p\_person\_password);  
**BEGIN  
 OPEN** person\_cursor;  
 **FETCH** person\_cursor **INTO** o\_person\_id, o\_person\_email, o\_person\_password, o\_person\_first\_name, o\_person\_last\_name, o\_person\_father\_name, o\_role\_name;  
 **IF** person\_cursor%**NOTFOUND THEN** *RAISE\_APPLICATION\_ERROR*(-20002, **'Login/Password error'**);  
 **END IF**;  
 **CLOSE** person\_cursor;  
**END** *log\_in\_person*;

Листинг 3.5 – Описание процедуры log\_in\_person

Параметр p\_person\_email проверяется для поиска данных клиента. В случае отсутствия пользователя с соответствующим адресом электронной почты возникает и обрабатывается ошибка. В случае успешной авторизации пользователю будет доступна информация об его аккаунте, окно для бронирования номеров, а также архив забронированных им номеров.

## **Процедура подтверждения брони администратором**

Для того, чтобы администратор мог контролировать созданные брони гостями, была создана процедура approve\_booking, позволяющая одобрять брони. Скрипт такой процедуры представлен в листинге 3.7.

**CREATE OR REPLACE PROCEDURE** *approve\_booking* (  
 p\_booking\_id **IN** booking\_table.**booking\_id**%**TYPE** )  
**IS** cnt **NUMBER**;  
**BEGIN  
 SELECT** *COUNT*(*\**) **INTO** cnt **FROM** booking\_table **WHERE booking\_id** = p\_booking\_id;  
 **IF** (cnt != 0) **THEN  
 UPDATE** booking\_table **SET booking\_state** = 1 **WHERE booking\_id** = p\_booking\_id;  
 **COMMIT**;  
 **ELSE** *RAISE\_APPLICATION\_ERROR*(-20023, **'Booking is not found'**);  
 **END IF**;  
**END** *approve\_booking*;

Листинг 3.7 – Описание процедуры approve\_booking

## **Предоставление и изменение информации о пользователе**

Информацию о пользователе (логин, пароль, адрес электронной почты, имя, фамилия, отчество) заполняет сам пользователь в окне регистрации. При авторизации данные об игроке можно увидеть в окне «Settings» в соответствующих полях. При необходимости любое из них может быть изменено, с помощью процедуры update\_person, представленной в листинге 3.8.

**CREATE OR REPLACE PROCEDURE** *update\_person* (  
 p\_person\_id **IN** person\_table.**person\_id**%**TYPE**,  
 p\_person\_email **IN** person\_table.**person\_email**%**TYPE**,  
 p\_person\_password **IN** person\_table.**person\_password**%**TYPE**,  
 p\_person\_first\_name **IN** person\_table.**person\_first\_name**%**TYPE**,  
 p\_person\_last\_name **IN** person\_table.**person\_last\_name**%**TYPE**,  
 p\_person\_father\_name **IN** person\_table.**person\_father\_name**%**TYPE** )  
**IS** cnt **NUMBER**;  
**BEGIN  
 SELECT** *COUNT*(*\**) **INTO** cnt **FROM** person\_table **WHERE person\_id** = p\_person\_id;  
 **IF** (cnt != 0) **THEN  
 SELECT** *COUNT*(*\**) **INTO** cnt **FROM** person\_table **WHERE** *UPPER*(**person\_email**) = *UPPER*(p\_person\_email) **AND person\_id** != p\_person\_id;  
 **IF** (cnt = 0) **THEN  
 UPDATE** person\_table **SET person\_email** = p\_person\_email, **person\_password** = *encryption\_password*(p\_person\_password), **person\_first\_name** = p\_person\_first\_name,   
 **person\_last\_name** = p\_person\_last\_name, **person\_father\_name** = p\_person\_father\_name **WHERE person\_id** = p\_person\_id;  
 **COMMIT**;  
 **ELSE** *RAISE\_APPLICATION\_ERROR*(-20001, **'This person already exists'**);  
 **END IF**;  
 **ELSE** *RAISE\_APPLICATION\_ERROR*(-20003, **'Person is not found'**);  
 **END IF**;  
**END** *update\_person*;

Листинг 3.8 – Описание процедуры update\_person

Процедура поиска информации о госте по его адресу электронной почты представлена в листинге 3.9.

**CREATE OR REPLACE PROCEDURE** *search\_person* (  
 p\_person\_email **IN** person\_table.**person\_email**%**TYPE**,  
 o\_person\_id **OUT** person\_table.**person\_id**%**TYPE**,

Продолжение листинга 3.9

o\_person\_email **OUT** person\_table.**person\_email**%**TYPE**,  
 o\_person\_first\_name **OUT** person\_table.**person\_first\_name**%**TYPE**,  
 o\_person\_last\_name **OUT** person\_table.**person\_last\_name**%**TYPE**,  
 o\_person\_father\_name **OUT** person\_table.**person\_father\_name**%**TYPE** )  
**IS  
 CURSOR** person\_cursor **IS SELECT person\_id**, **person\_email**, **person\_first\_name**, **person\_last\_name**, **person\_father\_name FROM** person\_table **WHERE** *UPPER*(**person\_email**) = *UPPER*(p\_person\_email);  
**BEGIN  
 OPEN** person\_cursor;  
 **FETCH** person\_cursor **INTO** o\_person\_id, o\_person\_email, o\_person\_first\_name, o\_person\_last\_name, o\_person\_father\_name;  
 **IF** person\_cursor%**NOTFOUND THEN** *RAISE\_APPLICATION\_ERROR*(-20003, **'Person is not found'**);  
 **END IF**;  
 **CLOSE** person\_cursor;  
**END** *search\_person*;

Листинг 3.9 – Описание процедуры search\_person

## **Пользователи**

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 было использовано 2 пользователя.

Первый пользователь – Guest – зарегистрированный пользователь, имеет доступ к своим личным данным, может просматривать список свободных номеров, предлагать для одобрения администратором брони, может менять личную информацию, просматривать свой архив забронированных номеров (листинг 3.10).

**CREATE USER** HotelGuest  
**IDENTIFIED BY** Pa$$w0rd   
**DEFAULT TABLESPACE Users   
QUOTA UNLIMITED ON Users**;

Листинг 3.10 – Скрипт создания пользователя HotelGuest

Второй пользователь – Administrator – обладает иным перечнем прав. Имеет права для чтения и создания новых комнат, модификации сервисов и инвентарей, может одобрять либо отклонять брони, изменять личные данные, вызывать процедуры экспорта и импорта, загружать и удалять 100000 строк в таблицу (листинг 3.11).

**CREATE USER** HotelAdmin  
**IDENTIFIED BY** Pa$$w0rd   
**DEFAULT TABLESPACE Users   
QUOTA UNLIMITED ON Users**;

Листинг 3.11 – Скрипт создания пользователя HotelAdmin

Скрипт выдачи прав пользователям на представления, процедуры и функции представлен в приложении Г.

## **Описание процедур экспорта и импорта**

Для экспорта таблицы inventory\_type\_table в формате xml была разработана процедура export\_inventory\_type. Для реализации этих процедур потребовалось указать директорию для сохранения xml файла, в который будет производиться импорт и экспорт. В нашем случае имя директории – «DATA\_PUMP\_DIR», путь – «/opt/oracle/admin/XE/dpdump/». Процедура export\_inventory\_type представлена ниже в листинге 4.1.

**CREATE OR REPLACE PROCEDURE** *export\_inventory\_type***IS** rc **sys\_refcursor**;  
 doc DBMS\_XMLDOM.DOMDocument;  
**BEGIN  
 OPEN** rc **FOR SELECT inventory\_type\_id**, **inventory\_type\_name FROM** HotelAdmin.inventory\_type\_table;  
 doc := DBMS\_XMLDOM.*NewDOMDocument*(**XMLTYPE**(rc));  
 DBMS\_XMLDOM.*WRITETOFILE*(doc, **'DATA\_PUMP\_DIR/export\_inventory\_type.xml'**);  
**END** *export\_inventory\_type*;

Листинг 4.1 – Скрипт создания процедуры export\_inventory\_type

Для выгрузки большого чиста строк создаем курсорную переменную rc. Для создания xml файла и экспорта в него потребуется использовать пакет DBMS\_XMLDOM. Объявляем переменную типа DOMDocument.

В блоке begin построчно считываем необходимые строки из таблицы и помещаем их в курсорную переменную. Создаем новый DOMDocument, содержимое которого – строки курсорной переменной. После чего при помощи функции WRITETOFILE записываем содержимое DOMDocument’а в созданный файл с именем export\_inventory\_type.xml

Для импорта данных в таблицу inventory\_type\_table из файла формата xml была разработана процедура import\_inventory\_type. Данная процедура представлена в листинге 4.2:

**CREATE OR REPLACE PROCEDURE** *import\_inventory\_type***IS  
BEGIN  
 INSERT INTO** inventory\_type\_table (**inventory\_type\_name**)  
 **SELECT** *ExtractValue*(*VALUE*(inventory\_type\_xml), **'//INVENTORY\_TYPE\_NAME'**) **AS** inventory\_type\_name

**FROM TABLE**(*XMLSequence*(*EXTRACT*(**XMLTYPE**(*bfilename*(**'DATA\_PUMP\_DIR'**, **'import\_inventory\_type.xml'**),

Продолжение листинга 4.2

*nls\_charset\_id*(**'UTF-8'**)),**'/ROWSET/ROW'**))) inventory\_type\_xml;  
**END** *import\_inventory\_type*;

Листинг 4.2 − Данные XML для импорта в базу данных

Указываем таблицу и столбцы, в которые будем производить вставку. При помощи функции ExtractValue даем установку, что записи с определенными тегами будут трактоваться как данные определенных столбцов. После чего при помощи той же функции указываем местоположение файла и его название, используемую кодировку и корень файла.

Таким образом мы можем быстро и удобно импортировать данные в нашу таблицу.

## **Описание и реализация технологий**

## **Шифрование данных**

В разработке своего проекта я использовала технологию шифрования данных. В Oracle для этого чаще всего используются следующие алгоритмы:

* DES (Data Encryption Standard). Традиционно алгоритм DES занимал ведущие позиции в области шифрования. Алгоритму DES необходим 64-разрядный ключ, но 8 бит ключа не используются. Чтобы подобрать ключ, злоумышленнику придется перебрать до 72 057 594 037 927 936 комбинаций.
* DES3. В этой схеме, базирующейся на исходном алгоритме DES, данные шифруются дважды или трижды (в зависимости от режима вызова). DES3 использует 128- или 192-разрядный ключ; его длина определяется количеством проходов.
* AES. В ноябре 2001 года был одобрен новый стандарт AES (Advanced Encryption Standard), вступивший в силу в мае 2002 года.

Блок данных обычно не шифруется как единое целое. Чаще всего он разбивается на фрагменты по 8 байт, после чего каждый фрагмент шифруется независимо от других. Конечно, длина данных может быть не кратной 8 — в этом случае алгоритм добавляет символы в последний фрагмент до 8 байт. Этот процесс называется заполнением (padding).

Шифрация данных происходит при помощи функции DBMS\_CRYPTO.ENCRYPT. Функция получает четыре аргумента:

* src — исходные данные, подлежащие шифрованию (должны иметь тип данных RAW).
* key — ключ шифрования (также RAW). Длина ключа должна соответствовать выбранному алгоритму. Например, для алгоритма DES она должна быть не менее 64 бит.
* typ — определение трех компонентов (алгоритм, механизм заполнения и метод сцепления) в виде суммы соответствующих констант.
* iv — необязательный вектор инициализации (IV), еще один компонент схемы шифрования, затрудняющий анализ «закономерностей» в зашифрованном тексте.

В своем приложении я использовала:

* алгоритм — AES c 128-разрядным ключом;
* метод сцепления — CBC;
* механизм заполнения — PKCS#5.

Они задаются следующим значением параметра typ при вызове функции:

l\_mod **NUMBER** := DBMS\_CRYPTO.encrypt\_aes128 + DBMS\_CRYPTO.chain\_cbc + DBMS\_CRYPTO.pad\_pkcs5;

Для шифрования необходимо выбрать ключ. В приложении в качестве ключа используется строка «2901200316052000». Это значение относится к типу данных VARCHAR2. Чтобы использовать его в функции ENCRYPT, необходимо сначала преобразовать его к типу RAW. Для этого мы воспользуемся функцией STRING\_TO\_RAW встроенного пакета UTL\_I18N. Также вторым параметром указываем кодировку.

Переменная l\_in\_val типа VARCHAR2 также должна быть преобразована к типу RAW. Переменную, которая будет передаваться в качестве возвращаемого значения функции, требуется преобразовать в шестнадцатеричный формат. Для этого используем функцию RAWTOHEX.

Процедура шифрования пароля представлена ниже в листинге 5.1:

**CREATE OR REPLACE FUNCTION** *encryption\_password* (  
 p\_person\_password **IN** person\_table.**person\_password**%**TYPE** )  
 **RETURN** person\_table.**person\_password**%**TYPE  
IS** l\_key **VARCHAR2**(2000) := **'2901200316052000'**;  
 l\_in\_val **VARCHAR2**(2000) := p\_person\_password;  
 l\_mod **NUMBER** := DBMS\_CRYPTO.encrypt\_aes128 + DBMS\_CRYPTO.chain\_cbc + DBMS\_CRYPTO.pad\_pkcs5;  
 l\_enc **RAW**(2000);  
**BEGIN** l\_enc := DBMS\_CRYPTO.*encrypt*(utl\_i18n.*string\_to\_raw*(l\_in\_val, **'AL32UTF8'**), l\_mod, utl\_i18n.*string\_to\_raw*(l\_key, **'AL32UTF8'**));  
**RETURN** *RAWTOHEX*(l\_enc);  
**END** *encryption\_password*;

Листинг 5.1 – Шифрование данных с помощью алгоритма AES128

Шифрование данных имеет смысл только в том случае, если зашифрованные данные в какой-то момент будут прочитаны и использованы в приложении. Эта задача решается при помощи функции DECRYPT. По структуре вызова она идентична функции ENCRYPT и получает те же четыре аргумента:

* src — зашифрованные данные.
* key — ключ, использованный для шифрования.
* typ — три компонента (алгоритм, механизм заполнения и метод сцепления), использованные при вызове ENCRYPT.
* iv — вектор инициализации, использованный при вызове ENCRYPT.

Функция DECRYPT тоже возвращает дешифрованные данные в формате RAW; для нормального просмотра их необходимо преобразовать в другой формат. Прежде всего преобразуем шестнадцатеричный формат зашифрованного пароля в RAW. Для этого воспользуемся функцией HEXTORAW.

Используя тот же самый ключ, передаем его в качестве одного из параметров в функцию DECRYPT, перед этим преобразовав его в RAW при помощи функции STRING\_TO\_RAW. Так же не забываем передать в функцию параметр typ и зашифрованный пароль типа RAW.

Результат функции преобразуем к строке при помощи функции RAW\_TO\_CHAR пакета UTL\_I18N.

Процедура дешифрования пароля представлена ниже в листинге 5.2:

**CREATE OR REPLACE FUNCTION** *decryption\_password* (  
 p\_person\_password **IN** person\_table.**person\_password**%**TYPE** )  
 **RETURN** person\_table.**person\_password**%**TYPE  
IS** l\_key **VARCHAR2**(2000) := **'2901200316052000'**;  
 l\_in\_val **RAW**(2000) := *HEXTORAW*(p\_person\_password);  
 l\_mod **NUMBER** := DBMS\_CRYPTO.encrypt\_aes128 + DBMS\_CRYPTO.chain\_cbc + DBMS\_CRYPTO.pad\_pkcs5;  
 l\_dec **RAW**(2000);  
**BEGIN** l\_dec := DBMS\_CRYPTO.*decrypt*(l\_in\_val, l\_mod, utl\_i18n.*string\_to\_raw*(l\_key, **'AL32UTF8'**));  
**RETURN** utl\_i18n.*raw\_to\_char*(l\_dec);  
**END** *decryption\_password*;

Листинг 5.2 – Дешифрование данных

## **Хранение мультимедийных типов данных**

Для работы с данными большого объёма СУБД Oracle предоставляет типы данных BLOB, CLOB, NCLOB и BFILE. Здесь LOB означает large object, или большой объект, и далее по тексту термины LOB и "большой объект" взаимозаменяемы. По сути, большой объект - это абстрактный тип для манипуляции данными большого объема внутри БД, а типы BLOB, CLOB, NCLOB и BFILE - его конкретные реализации.

В курсовом проекте технология мультимедийных типов данных отражена в виде хранимых изображений комнат, инвентарей и пользователей. Изображения хранятся в соответствующих полях каждой из этих таблиц (room\_table.room\_album\_id, inventory\_table.inventory\_album\_id, person\_table.person\_album\_id).

Для хранения в поле изображений в СУБД применяется тип BLOB. Тип BLOB представляет из себя массив двоичных данных. Пример процедуры, который позволяет хранить внутри таблицы изображение, приведен ниже:

**CREATE OR REPLACE PROCEDURE** *update\_photo\_source* (  
 p\_photo\_id **IN** photo\_table.photo\_id%**TYPE**,  
 p\_source\_name **IN NVARCHAR2** )  
**IS** src\_file **BFILE**;  
 dst\_file **BLOB**;  
 lgh\_file **BINARY\_INTEGER**;  
**BEGIN** src\_file := *bfilename*(**'DATA\_PUMP\_DIR'**, p\_source\_name);  
 **SELECT photo\_source INTO** dst\_file **FROM** photo\_table **WHERE photo\_id** = p\_photo\_id **FOR UPDATE**;  
 DBMS\_LOB.*FILEOPEN*(src\_file, DBMS\_LOB.FILE\_READONLY);  
 lgh\_file := DBMS\_LOB.*GETLENGTH*(src\_file);  
 DBMS\_LOB.*LOADFROMFILE*(dst\_file, src\_file, lgh\_file);  
 **UPDATE** photo\_table **SET photo\_source** = dst\_file **WHERE photo\_id** = p\_photo\_id;  
 DBMS\_LOB.*FILECLOSE*(src\_file);  
 **COMMIT**;  
**END** *update\_photo\_source*;

Листинг 5.3 – Процедура для хранения внутри таблицы изображения

Для начала требуется указать путь на статическую директорию, в нашем случае, это директория «DATA\_PUMP\_DIR». Она указывает на местоположение всех картинок.

При помощи функции DBMS\_LOB.FILEOPEN() мы открываем файл (в нашем случае изображение) в BFILE src\_file. В переменную lgh\_file мы помещаем при помощи функции DBMS\_LOB.GETLENGTH() длину этого файла в байтах.

Наконец, вставляем в переменную dst\_file типа BLOB, используя функцию DBMS\_LOB.LOADFROMFILE(), необходимый файл, который хранится в src\_file, указывая объем этого файла переменной lgh\_file.

Используя оператор UPDATE, заменяем содержимое BLOB столбца по имени, указанному в качестве параметра процедуры, после чего закрываем файл при помощи функции DBMS\_LOB.FILECLOSE().



## **Тестирование производительности**

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса. К таблице role\_table была разработана процедура insert\_100k для заполнения таблицы строками в количестве 100000, представленная ниже в листинге 6.1:

**CREATE OR REPLACE PROCEDURE** *insert\_100k***IS  
BEGIN  
 FOR** i **IN** 1 .. 100000 **LOOP  
 INSERT INTO** role\_table(**role\_name**) **VALUES**(**'100k role'**);  
 **END LOOP**;  
 **COMMIT**;  
**END** *insert\_100k*;

Листинг 6.1 – Описание процедуры insert\_100k

Таким образом было добавлено 100000 строк в таблицу. Примерное время выполнения данной процедуры – 5 секунд (представлено на рисунке 6.1).



Рисунок 6.1 – Время выполнения процедуры insert\_100k

Для удаление вставленных данных в количестве 100000 строк, была также разработана процедура delete\_100k, представленная в листинге 6.1:

**CREATE OR REPLACE PROCEDURE** *delete\_100k***IS  
BEGIN  
 DELETE FROM** role\_table **WHERE role\_id** > 3;  
 **COMMIT**;  
**END** *delete\_100k*;

Листинг 6.2 – Описание процедуры delete\_100k

Примерное время выполнения данной процедуры – 5 секунд (представлено на рисунке 6.2).



Рисунок 6.2 – Время выполнения процедуры delete\_100k

Тем самым мы создаём разнообразие при случайном заполнении таблицы на довольно-таки большое количество строк.

## **Руководство пользователя по работе с программным средством**

При запуске приложения открывается окно авторизации, где можно ввести электронную почту и пароль. Если же аккаунта пользователя ещё нет, можно создать новый, нажав на кнопку «Registration».

Рисунок 7.1 − Окно авторизации

Если вы нажали на кнопку регистрации, откроется окно, где вам необходимо заполнить поля, необходимые для регистрации пользователя, а именно: email, пароль, имя, фамилия, отчество. Если вам необходимо вернуться к окну авторизации без регистрации нового аккаунта, вы можете воспользоваться кнопкой «Back to log in».

Рисунок 7.2 − Окно регистрации

После успешной регистрации, пользователь авторизуется и попадает на главную страницу программного средства для бронирования номера. Кроме этого, существуют ещё два окна, доступные пользователю: архив и меню настроек.

Для поиска доступных номеров пользователю необходимо задать несколько фильтров, такие как тип комнаты, диапазон предпочитаемой стоимости номера, дата заезда и дата выезда. При нажатии на кнопку «Find», появятся доступные номера согласно критериям, которые указал пользователь.

Рисунок 7.3 − Поиск необходимого номера

Двойным щелчком мыши по номеру мы переходим в окно для бронирования, там указываются дополнительные жильцы номера (не распространяется на одноместный номер), дополнительный инвентарь в номер можно добавить, щёлкнув на «+» в разделе «Rent», в котором можно выбрать тип инвентаря, название и дату аренды. Далее при бронировании можно дополнительно заказать обслуживающий сервис в разделе «Subscription», указав тип сервиса, выбрав электронную почту работника и дату подписки на данный сервис. Сверив всю информацию, указанную при бронировании, пользователь может нажать на кнопку «Book», после чего данная бронь будет зарегистрирована.

Автоматически пользователь перенаправляется на страницу «Archive», где может просмотреть действующие брони и информацию о них или отменить бронь до подтверждения администратором, нажав для этого на кнопку «Cancel booking».

Рисунок 7.4 − Архив пользователя

На странице настроек пользователь может изменить любую информацию о себе. Нажав на кнопку «Change person info», поля можно будет изменить. После внесённых изменений пользователю останется лишь нажать на кнопку «Save person info» для сохранения изменений в базе данных или «Revert person changes» для отката изменений, внесенных пользователем в данный момент. В этом же окне можно удалить аккаунт пользователя, вошедшего в систему, нажав на кнопку «Delete person» или выйти из аккаунта с помощью кнопки «Log out».

Рисунок 7.5 − Окно настроек информации о пользователе

При входе в приложение работника отеля ему, кроме страниц, описанных выше у обычного гостя отеля, доступна страница «Services» для отображения номеров, где он нанят на работу.

Рисунок 7.6 − Страница «Services» работника отеля

Что касается администратора базы данных, то ему доступно гораздо больше функций в шести страницах программного средства: Booking, People, Rooms, Inventories, Services, Settings.

На первой странице бронирования ему отображается вся информация о бронях гостей и работников, которые он может одобрить, нажав на кнопку «Approve booking» или удалить, нажав на кнопку «Cancel booking».

Рисунок 7.7 − Информация о зарегистрированных бронях

На второй странице администратору доступен список зарегистрированных в базе данных персон, которых он может назначать в качестве гостя или работника отеля.

Рисунок 7.8 − Список зарегистрированных пользователей

Третья страница предназначена для создания, обновления и удаления типов комнат и самих комнат. Щёлкнув на любую комнату из списка справа, мы можем изменить её информацию, кроме id, который назначается комнате автоматически при её создании.

Рисунок 7.9 − Создание, обновление, удаление комнат

Четвёртая страница администратора служит для создания, обновления, удаления инвентарей и их типов. Структура страницы аналогична третьей странице.

Рисунок 7.10 − Создание, обновление, удаление инвентарей

Пятая страница предназначена для создания, обновления и удаления из базы данных сервисов и их типов.

Рисунок 7.11 − Создание, обновление, удаление сервисов

Последняя страница администратора «Settings» схожа с такой же страницей гостя, но в данном случае здесь присутствуют дополнительные кнопки для тестирования процедур в базе данных: «export inventory type» - вызов процедуры базы данных для экспорта данных из таблицы INVENTORY\_TYPE\_TABLE, «import inventory type» - вызов процедуры для импорта данных в таблицу INVENTORY\_TYPE\_TABLE, «Insert 100k roles» - вызов процедуры для вставки 100000 строк в таблицу ROLE\_TABLE, «Delete 100k roles» - вызов процедуры для удаления 100000 строк из таблицы ROLE\_TABLE.

Рисунок 7.12 − Окно настроек информации об администраторе

## **Заключение**

В ходе выполнения курсовой работы была спроектирована база данных для работы отеля на базе СУБД Oracle 12c. Приложение было разработано с использованием технологии WPF .Net. Во время выполнения было изучено немало теоретического материала, а также просмотрено и разобрано большое количество уже готовых решений тех или иных задач. Были разработаны, описаны и применены на практике процедуры экспорта и импорта данных; разработаны функции шифрования и дешифрования данных пользователей; было описано руководство пользователя для приложения.

Функционально были выполнены следующие задачи:

* + - администратор одобряет новые брони, поступающие от клиентов;
    - администратор добавляет новые номера, инвентари и сервисы;
    - администратор просматривает всех зарегистрированных пользователей и может назначать им роли;
    - администратор и клиент имеют возможность изменять личные данные;
    - клиент просматривает доступные номера по выбранным фильтрам;
    - клиент может заказать дополнительные услуги в номер при регистрации брони;
    - клиенту доступен архив с информацией о его бронях;
    - были реализованы технологии шифрования на базе DBMS\_CRYPTO,

а также хранения мультимедийных данных на базе DBMS\_LOB.

Данное программное средство имеет удобный и понятный пользователю интерфейс, корректно обрабатывает ошибочный ввод данных (например, ввод данных некорректного формата, попытка оставить поля пустыми), а также некоторые внутренние ошибки.

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

## **Список использованной литературы**

[1] METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: https://metanit.com Дата доступа: 10.12.2019

[2] Oracle-dba.ru [Электронный ресурс] / Режим доступа: https://oracle-dba.ru Дата доступа 10.12.2019

[3] stackoverflow.com [Электронный ресурс] / Режим доступа: https://stackoverflow.com Дата доступа: 12.12.2019

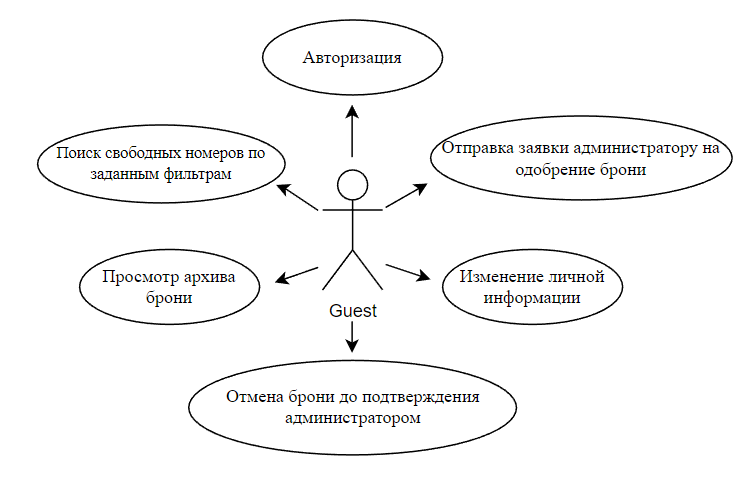
[4] Шифрование в PL/SQL: уроки программирования [Электронный ресурс] / Режим доступа: <https://oracle-patches.com/db/sql/3939-shifrovanie-v-pl-sql-uroki-programmirovaniya> Дата доступа: 13.12.2019

[5] DBMS\_SCHEDULER [Электронный ресурс] / Режим доступа: <https://docs.oracle.com/database/121/ARPLS/d_sched.htm#ARPLS72235> Дата доступа: 13.12.2019

[6] DBMS\_LOB [Электронный ресурс] / Режим доступа: <https://docs.oracle.com/database/121/ARPLS/d_lob.htm#ARPLS600> Дата доступа: 13.12.2019

[7] DBMS\_XMLDOM [Электронный ресурс] / Режим доступа: <https://docs.oracle.com/cd/B19306_01/appdev.102/b14258/d_xmldom.htm#i1076719>Дата доступа: 03.12.2019.

## **Приложение A**





## **Приложение Б**

|  |
| --- |
| *-----------------------------ROLE------------------------------*  **CREATE TABLE** role\_table (  **role\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **role\_name NVARCHAR2**(50) **NOT NULL**,  **CONSTRAINT** role\_pk **PRIMARY KEY** (**role\_id**) );  *----------------------------ALBUM------------------------------*  **CREATE TABLE** album\_table (  **album\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **album\_name NVARCHAR2**(50) **NOT NULL**,  **CONSTRAINT** album\_pk **PRIMARY KEY** (**album\_id**) );  *----------------------------PHOTO------------------------------*  **CREATE TABLE** photo\_table (  **photo\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **photo\_album\_id NUMBER**(10) **NOT NULL**,  **photo\_source BLOB DEFAULT** *EMPTY\_BLOB*(),  **CONSTRAINT** photo\_pk **PRIMARY KEY** (**photo\_id**),  **CONSTRAINT** photo\_album\_fk **FOREIGN KEY** (**photo\_album\_id**) **REFERENCES** album\_table(**album\_id**) );  *---------------------------PERSON------------------------------*  **CREATE TABLE** person\_table (  **person\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **person\_role\_id NUMBER**(10) **DEFAULT** 3 **NOT NULL**,  **person\_album\_id NUMBER**(10),  **person\_email NVARCHAR2**(50) **NOT NULL**,  **person\_password NVARCHAR2**(50) **NOT NULL**,  **person\_first\_name NVARCHAR2**(50) **NOT NULL**,  **person\_last\_name NVARCHAR2**(50) **NOT NULL**,  **person\_father\_name NVARCHAR2**(50) **NOT NULL**,  **CONSTRAINT** person\_pk **PRIMARY KEY** (**person\_id**),  **CONSTRAINT** person\_role\_fk **FOREIGN KEY** (**person\_role\_id**) **REFERENCES** role\_table(**role\_id**),  **CONSTRAINT** person\_album\_fk **FOREIGN KEY** (**person\_album\_id**) **REFERENCES** album\_table(**album\_id**) );  *--------------------------ROOM\_TYPE----------------------------*  **CREATE TABLE** room\_type\_table (  **room\_type\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **room\_type\_name NVARCHAR2**(50) **NOT NULL**,  **room\_type\_capacity NUMBER**(10),  **room\_type\_daily\_price FLOAT**(10) **NOT NULL**,  **CONSTRAINT** room\_type\_pk **PRIMARY KEY** (**room\_type\_id**) );  *-----------------------------ROOM------------------------------*  **CREATE TABLE** room\_table (  **room\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **room\_room\_type\_id NUMBER**(10) **NOT NULL**,  **room\_album\_id NUMBER**(10),  **room\_number NVARCHAR2**(50) **NOT NULL**,  **room\_description NVARCHAR2**(200),  **CONSTRAINT** room\_pk **PRIMARY KEY** (**room\_id**),  **CONSTRAINT** room\_room\_type\_fk **FOREIGN KEY** (**room\_room\_type\_id**) **REFERENCES** room\_type\_table(**room\_type\_id**),  **CONSTRAINT** room\_album\_fk **FOREIGN KEY** (**room\_album\_id**) **REFERENCES** album\_table(**album\_id**) );  *------------------------INVENTORY\_TYPE-------------------------*  **CREATE TABLE** inventory\_type\_table (  **inventory\_type\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **inventory\_type\_name NVARCHAR2**(50) **NOT NULL**,  **CONSTRAINT** inventory\_type\_pk **PRIMARY KEY** (**inventory\_type\_id**) );  *---------------------------INVENTORY---------------------------* **CREATE TABLE** inventory\_table (  **inventory\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **inventory\_inventory\_type\_id NUMBER**(10) **NOT NULL**,  **inventory\_album\_id NUMBER**(10),  **inventory\_description NVARCHAR2**(200) **NOT NULL**,  **inventory\_daily\_price FLOAT**(10) **NOT NULL**,  **CONSTRAINT** inventory\_pk **PRIMARY KEY** (**inventory\_id**),  **CONSTRAINT** inventory\_inventory\_type\_fk **FOREIGN KEY** (**inventory\_inventory\_type\_id**) **REFERENCES** inventory\_type\_table(**inventory\_type\_id**),  **CONSTRAINT** inventory\_album\_fk **FOREIGN KEY** (**inventory\_album\_id**) **REFERENCES** album\_table(**album\_id**) );  *-------------------------SERVICE\_TYPE--------------------------*  **CREATE TABLE** service\_type\_table (  **service\_type\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **service\_type\_name NVARCHAR2**(50) **NOT NULL**,  **service\_type\_daily\_price FLOAT**(10) **NOT NULL**,  **CONSTRAINT** service\_type\_pk **PRIMARY KEY** (**service\_type\_id**) );  *----------------------------SERVICE----------------------------*  **CREATE TABLE** service\_table (  **service\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **service\_service\_type\_id NUMBER**(10) **NOT NULL**,  **service\_person\_id NUMBER**(10) **NOT NULL**,  **CONSTRAINT** service\_pk **PRIMARY KEY** (**service\_id**),  **CONSTRAINT** service\_service\_type\_fk **FOREIGN KEY** (**service\_service\_type\_id**) **REFERENCES** service\_type\_table(**service\_type\_id**),  **CONSTRAINT** service\_person\_fk **FOREIGN KEY** (**service\_person\_id**) **REFERENCES** person\_table(**person\_id**) );  *----------------------------BOOKING----------------------------*  **CREATE TABLE** booking\_table (  **booking\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **booking\_room\_id NUMBER**(10) **NOT NULL**,  **booking\_start\_date DATE NOT NULL**,  **booking\_end\_date DATE NOT NULL**,  **booking\_state NUMBER**(1) **DEFAULT** 0,  **CONSTRAINT** booking\_pk **PRIMARY KEY** (**booking\_id**),  **CONSTRAINT** booking\_room\_fk **FOREIGN KEY** (**booking\_room\_id**) **REFERENCES** room\_table(**room\_id**) ); *--booking\_state: --0 - booked by guest --1 - approved by the administrator*  *----------------------------RESIDENT---------------------------*  **CREATE TABLE** resident\_table (  **resident\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **resident\_person\_id NUMBER**(10) **NOT NULL**,  **resident\_booking\_id NUMBER**(10) **NOT NULL**,  **CONSTRAINT** resident\_pk **PRIMARY KEY** (**resident\_id**),  **CONSTRAINT** resident\_person\_fk **FOREIGN KEY** (**resident\_person\_id**) **REFERENCES** person\_table(**person\_id**),  **CONSTRAINT** resident\_booking\_fk **FOREIGN KEY** (**resident\_booking\_id**) **REFERENCES** booking\_table(**booking\_id**) );  *------------------------------RENT-----------------------------*  **CREATE TABLE** rent\_table (  **rent\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **rent\_inventory\_id NUMBER**(10) **NOT NULL**,  **rent\_booking\_id NUMBER**(10) **NOT NULL**,  **rent\_start\_date DATE NOT NULL**,  **rent\_end\_date DATE NOT NULL**,  **CONSTRAINT** rent\_pk **PRIMARY KEY** (**rent\_id**),  **CONSTRAINT** rent\_inventory\_fk **FOREIGN KEY** (**rent\_inventory\_id**) **REFERENCES** inventory\_table(**inventory\_id**),  **CONSTRAINT** rent\_booking\_fk **FOREIGN KEY** (**rent\_booking\_id**) **REFERENCES** booking\_table(**booking\_id**) );  *------------------------SUBSCRIPTION---------------------------*  **CREATE TABLE** subscription\_table (  **subscription\_id NUMBER**(10) **GENERATED AS IDENTITY**(**START WITH** 1 **INCREMENT BY** 1),  **subscription\_service\_id NUMBER**(10) **NOT NULL**,  **subscription\_booking\_id NUMBER**(10) **NOT NULL**,  **subscription\_start\_date DATE NOT NULL**,  **subscription\_end\_date DATE NOT NULL**,  **CONSTRAINT** subscription\_pk **PRIMARY KEY** (**subscription\_id**),  **CONSTRAINT** subscription\_service\_fk **FOREIGN KEY** (**subscription\_service\_id**) **REFERENCES** service\_table(**service\_id**),  **CONSTRAINT** subscription\_booking\_fk **FOREIGN KEY** (**subscription\_booking\_id**) **REFERENCES** booking\_table(**booking\_id**) ); |

## **Приложение В**

|  |  |  |
| --- | --- | --- |
| Название процедуры | Описание | Принимаемые параметры |
| insert\_100k | Процедура заполнения таблицы role\_table на 100000 строк |  |
| delete\_100k | Процедура удаления 100000 строк из таблицы role\_table |  |
| register\_person | Регистрация нового пользователя | p\_person\_email, p\_person\_password, p\_person\_first\_name, p\_person\_last\_name, p\_person\_father\_name |
| log\_in\_person | Авторизация пользователя | p\_person\_email, p\_person\_password,  o\_person\_id, o\_person\_email, o\_person\_password, o\_person\_first\_name, o\_person\_last\_name, o\_person\_father\_name, o\_role\_name |
| search\_person | Поиск пользователя | p\_person\_email, o\_person\_email, o\_person\_id, o\_person\_first\_name, o\_person\_last\_name, o\_person\_father\_name |
| update\_person | Обновление информации о пользователе | p\_person\_id, p\_person\_email, p\_person\_password, p\_person\_first\_name, p\_person\_last\_name, p\_person\_father\_name |
| delete\_person | Удаление пользователя | p\_person\_id |
| update\_role | Изменение роли пользователя | p\_person\_id, p\_role\_id |
| create\_room\_type | Добавление типа комнат | p\_room\_type\_name, p\_room\_type\_daily\_price |
| update\_room\_type | Обновление типа комнаты | p\_room\_type\_id, p\_room\_type\_name, p\_room\_type\_daily\_price |
| delete\_room\_type | Удаление типа комнаты | p\_room\_type\_id |
| create\_room | Добавление комнаты | p\_room\_room\_type\_id, p\_room\_number |
| update\_room | Обновление комнаты | p\_room\_id, p\_room\_room\_type\_id, p\_room\_number |
| delete\_room | Удаление комнаты | p\_room\_id |
| create\_inventory\_type | Создание типа инвентаря | p\_inventory\_type\_name |
| update\_inventory\_type | Обновление типа инвентаря | p\_inventory\_type\_id, p\_inventory\_type\_name |
| delete\_inventory\_type | Удаление типа инвентаря | p\_inventory\_type\_id |
| create\_inventory | Создание инвентаря | p\_inventory\_inventory\_type\_id, p\_inventory\_description, p\_inventory\_daily\_price |
| update\_inventory | Обновление инвентаря | p\_inventory\_id, p\_inventory\_inventory\_type\_id, p\_inventory\_description, p\_inventory\_daily\_price |
| delete\_inventory | Удаление инвентаря | p\_inventory\_id |
| create\_service\_type | Создание типа сервиса | p\_service\_type\_name, p\_service\_type\_daily\_price |
| update\_service\_type | Обновление типа сервиса | p\_service\_type\_id, p\_service\_type\_name, p\_service\_type\_daily\_price |
| delete\_service\_type | Удаление типа сервиса | p\_service\_type\_id |
| create\_service | Добавление сервиса | p\_service\_service\_type\_id, p\_service\_person\_id |
| update\_service | Обновление сервиса | p\_service\_id, p\_service\_service\_type\_id, p\_service\_person\_id |
| delete\_service | Удаление сервиса | p\_service\_id |
| create\_booking | Создание брони | p\_booking\_room\_id, p\_booking\_start\_date,  p\_booking\_end \_date |
| guest\_create\_booking | Создание гостем брони номера | p\_booking\_room\_id, p\_booking\_start\_date,  p\_booking\_end \_date, o\_booking\_id |
| update\_booking | Обновление брони | p\_booking\_id,  p\_booking\_room\_id,  p\_booking\_start\_date,  p\_booking\_end \_date |
| approve\_booking | Подтверждение брони администратором | p\_booking\_id |
| delete\_booking | Удаление брони | p\_booking\_id |
| guest\_delete\_booking | Удаление брони гостем | p\_booking\_id |
| create\_resident | Добавление жильцов номера | p\_resident\_person\_id, p\_resident\_booking\_id |
| update\_resident | Обновление информации о жильцах номера | p\_resident\_id,  p\_resident\_person\_id, p\_resident\_booking\_id |
| delete\_resident | Удаление жильцов | p\_resident\_id |
| create\_rent | Создание аренды инвентаря | p\_rent\_inventory\_id, p\_rent\_booking\_id,  p\_rent\_start\_date,  p\_rent\_end\_date |
| update\_rent | Обновление аренды | p\_rent\_id,  p\_rent\_inventory\_id, p\_rent\_booking\_id,  p\_rent\_start\_date,  p\_rent\_end\_date |
| delete\_rent | Удаление аренды | p\_rent\_id |
| create\_subscription | Создание подписки на сервис | p\_subscription\_service\_id, p\_subscription\_booking\_id, p\_subscription\_start\_date, p\_subscription\_end\_date |
| update\_subscription | Обновление информации о подписке на сервис | p\_subscription\_id, p\_subscription\_service\_id, p\_subscription\_booking\_id, p\_subscription\_start\_date, p\_subscription\_end\_date |
| delete\_subscription | Удаление подписки | p\_subscription\_id |
| create\_album | Создание альбома | p\_album\_name, p\_album\_object, o\_album\_id |
| update\_album | Обновление альбома | p\_album\_id, p\_album\_name, p\_album\_object |
| delete\_album | Удаление альбома | p\_album\_id |
| create\_photo | Создание фото | p\_photo\_album\_id |
| update\_photo | Обновление фото | p\_photo\_id, p\_photo\_album\_id |
| delete\_photo | Удаление фото | p\_photo\_id |
| update\_photo\_source | Обновление ресурса фото | p\_photo\_id, p\_source\_name |
| export\_inventory\_type | Экспорт данных |  |
| import\_inventory\_type | Импорт данных |  |

## **Приложение Г**

Скрипт распределения доступа по ролям:

|  |
| --- |
| *-------------------------HOTELADMIN----------------------------*  **GRANT CONNECT TO** HotelAdmin; **GRANT CREATE TABLE TO** HotelAdmin; **GRANT CREATE SEQUENCE TO** HotelAdmin; **GRANT CREATE VIEW TO** HotelAdmin; **GRANT CREATE INDEXTYPE TO** HotelAdmin; **GRANT CREATE PROCEDURE TO** HotelAdmin; **GRANT CREATE TRIGGER TO** HotelAdmin; **GRANT CREATE SESSION TO** HotelAdmin;  **GRANT EXECUTE ON sys**.dbms\_crypto **TO** HotelAdmin;  *-------------------------HOTELGUEST----------------------------*  **GRANT CREATE SESSION TO** HotelGuest;  **GRANT SELECT ON** HotelAdmin.room\_type\_table **TO** HotelGuest; **GRANT SELECT ON** HotelAdmin.inventory\_type\_table **TO** HotelGuest; **GRANT SELECT ON** HotelAdmin.service\_type\_table **TO** HotelGuest;  **GRANT SELECT ON** HotelAdmin.album\_view **TO** HotelGuest; **GRANT SELECT ON** HotelAdmin.room\_view **TO** HotelGuest; **GRANT SELECT ON** HotelAdmin.inventory\_view **TO** HotelGuest; **GRANT SELECT ON** HotelAdmin.service\_view **TO** HotelGuest; **GRANT SELECT ON** HotelAdmin.booking\_view **TO** HotelGuest; **GRANT SELECT ON** HotelAdmin.resident\_view **TO** HotelGuest; **GRANT SELECT ON** HotelAdmin.rent\_view **TO** HotelGuest; **GRANT SELECT ON** HotelAdmin.subscription\_view **TO** HotelGuest;  **GRANT EXECUTE ON** HotelAdmin.*create\_album* **TO** HotelGuest; **GRANT EXECUTE ON** HotelAdmin.*delete\_album* **TO** HotelGuest; **GRANT EXECUTE ON** HotelAdmin.*register\_person* **TO** HotelGuest; **GRANT EXECUTE ON** HotelAdmin.*log\_in\_person* **TO** HotelGuest; **GRANT EXECUTE ON** HotelAdmin.*search\_person* **TO** HotelGuest; **GRANT EXECUTE ON** HotelAdmin.*update\_person* **TO** HotelGuest; **GRANT EXECUTE ON** HotelAdmin.*delete\_person* **TO** HotelGuest; **GRANT EXECUTE ON** HotelAdmin.*encryption\_password* **TO** HotelGuest; **GRANT EXECUTE ON** HotelAdmin.*decryption\_password* **TO** HotelGuest; **GRANT EXECUTE ON** HotelAdmin.*guest\_create\_booking* **TO** HotelGuest; **GRANT EXECUTE ON** HotelAdmin.*create\_resident* **TO** HotelGuest; **GRANT EXECUTE ON** HotelAdmin.*create\_rent* **TO** HotelGuest; **GRANT EXECUTE ON** HotelAdmin.*create\_subscription* **TO** HotelGuest; **GRANT EXECUTE ON** HotelAdmin.*guest\_delete\_booking* **TO** HotelGuest; |