МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема: «Разработка базы данных туристического агентства с использованием средств мониторинга состояния СУБД»

**Исполнитель**

студент 2 курса 2 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. И. Дрозд

подпись, дата

**Руководитель**

ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В.Харланович

(должность, уч. звание) (подпись, дата)

Допущен(а) к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Харланович

подпись дата инициалы и фамилия

Минск 2022

**Оглавление**

[Введение 4](#_Toc134955827)

[1 Анализ требований к создаваемому продукту 5](#_Toc134955828)

[1.1. Аналитический обзор аналогов 5](#_Toc134955829)

[1.2. Разработка функциональных требований, определение вариантов использования 7](#_Toc134955830)

[1.3 Вывод 9](#_Toc134955831)

[2 Разработка архитектуры проекта 10](#_Toc134955832)

[2.1. Обобщенная структура управлением приложения 10](#_Toc134955833)

[2.2. Диаграммы UML, взаимосвязь всех компонентов. 10](#_Toc134955834)

[2.3. Описание информационных объектов 11](#_Toc134955835)

[2.4. Вывод 12](#_Toc134955836)

[3 Разработка модели базы данных 13](#_Toc134955837)

[3.1. Создание необходимых объектов 13](#_Toc134955838)

[3.1.1. Представления базы данных 13](#_Toc134955839)

[3.1.2. Индексы базы данных 14](#_Toc134955840)

[3.1.3. Триггеры базы данных 15](#_Toc134955841)

[3.2. Описание используемой технологии 16](#_Toc134955842)

[3.3. Вывод 16](#_Toc134955843)

[4 Установка, настройка и использование Oracle 12c 18](#_Toc134955844)

[4.1. Установка Oracle 12c 18](#_Toc134955845)

[4.2. Создание таблиц 18](#_Toc134955846)

[4.3. Создание ролей для разграничения 19](#_Toc134955847)

[4.4. Создание пакетов процедур для базы данных 21](#_Toc134955848)

[4.4.1. Выборка данных из таблиц 22](#_Toc134955849)

[4.4.2. Заполнение таблиц 100 000 строк 23](#_Toc134955850)

[4.4.3. Удаление данных в таблицы 23](#_Toc134955851)

[4.4.4. Добавление данных в таблицы 24](#_Toc134955852)

[4.4.5. Экспорт и импорт данных 24](#_Toc134955853)

[4.5. Вывод 26](#_Toc134955854)

[5 Тестирование 27](#_Toc134955855)

[5.1. Тестирование производительности базы данных 27](#_Toc134955856)

[5.2. Вывод 28](#_Toc134955857)

[6 Руководство использования программного средства 29](#_Toc134955858)

[6.1. Руководство пользователя 29](#_Toc134955859)

[6.2. Вывод 31](#_Toc134955860)

[Заключение 32](#_Toc134955861)

[Список литературных источников 33](#_Toc134955862)

[Приложение А 34](#_Toc134955863)

# Введение

Целью данной курсовой работы является создание реляционной базы данных для туристического агентства, которая предоставляет пользователям доступ к путевкам, предлагаемым агентством.

Рынок туристических услуг в настоящее время стремительно развивается. Причиной этому являются не только растущие доходы и повышенный интерес к путешествиям, но и технологическое развитие. Сегодня технологии позволяют туристическим агентствам предоставлять более удобные и быстрые сервисы, снижать затраты и повышать качество обслуживания клиентов.

Одним из наиболее важных направлений технологического развития рынка туристических услуг является разработка и использование баз данных и информационных систем. БД и ИС позволяют туристическим агентствам эффективно управлять информацией о клиентах, турах, отелях и других услугах, быстро анализировать данные и принимать правильные решения. Технологии аналитики и искусственного интеллекта также помогают туристическим агентствам улучшить сервис и предложить клиентам наиболее подходящие услуги.

База данных — это организованное собрание данных, которое обычно хранится в электронном виде в компьютерной системе. БД используются для хранения, организации и управления большим объемом структурированных и неструктурированных данных. Реляционная база данных является наиболее распространенной формой организации данных, в которой данные представлены в виде таблиц, состоящих из строк и столбцов, где каждый столбец представляет атрибут, а каждая строка представляет кортеж или запись. В данной работе для управления базой данных была выбрана СУБД Oracle 12c, поскольку эта система обладает высокой надежностью и производительностью, что позволяет обеспечить эффективное хранение, обработку и управление данными.

Основными задачами моего проекта являются:

* Создание ролей (пользователь, менеджер, администратор);
* Реализация поиска туров по параметрам;
* Управление туристическими турами и ваучерами (добавление, удаление, изменение);
* Создание запросов анализа работы предприятия.

1. Анализ требований к создаваемому продукту
   1. Аналитический обзор аналогов

Путешествия — это одно из самых захватывающих приключений в жизни. Каждый раз, когда мы отправляемся в путешествие, мы получаем незабываемые впечатления, расширяем свой кругозор, насыщаем себя новыми знаниями и узнаем много интересного о мире, в котором мы живем.

Путешествия могут быть разнообразными - от коротких выходных на природе до масштабных мировых туров. Независимо от того, где вы едете и сколько времени проводите в пути, путешествие всегда представляет собой возможность вырваться из обыденности и погрузиться в новые впечатления.

Кроме того, путешествия могут быть полезны не только для души, но и для физического и психического здоровья. Они позволяют расслабиться, отвлечься от стрессов и проблем, а также поддерживать физическую форму за счет активного отдыха и разнообразной деятельности.

Одним из основных преимуществ туристических фирм является возможность приобретения готового тура в интересующую страну без необходимости самостоятельно организовывать поездку (бронирование отеля, поиск билетов на самолет и т. д.). Это делает туристические агентства очень востребованными среди большинства людей по всему миру.

Ниже приведены аналоги туристических агентств.

Туристическая компания АлатанТур успешно работает на рынке с 1993 года и является одной из крупнейших в Беларуси. На данный момент АлатанТур располагает тремя офисами в Минске, а также семью офисами в крупных городах Республики Беларусь. Пример интерфейса данного сервиса представлен на рисунке 1.1.

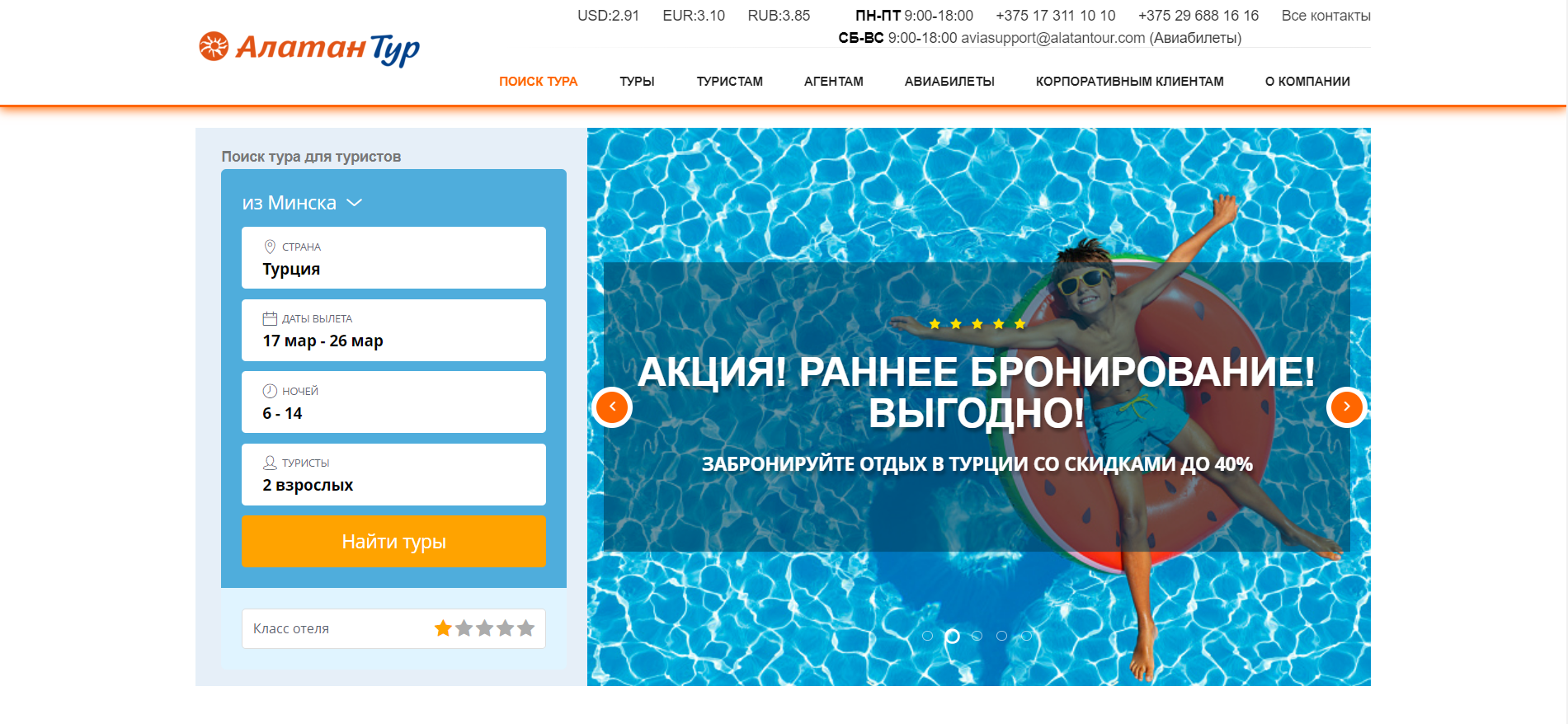


Рисунок 1.1 – Интерфейс сервиса АталанТур

Одной из основных функций, предложенных на сервисе АталанТур является поиск организованных туров. Пользователи могут искать интересующий тур исходя из предложенных агентством.

Также пользователи могут сами указывать необходимые параметры для создания путевки. Для этого им необходимо заполнить всю информацию о создаваемом туре: выбор страны, выбор качества отеля, выбор города проживания, указание интересующих дат и прочее.

Дополнительно АталанТур предоставляет возможность покупки авиабилетов для юридических лиц. При выборе билетов клиент самостоятельно указывает необходимую дату, время, начальную и конечную точку маршрута.

Еще одним из аналогов туристического агентства, рассмотренного в ходе обзора, является CORAL TRAVEL. CORAL TRAVEL предлагает лучшие курорты и отели в 30 странах мира, среди которых – Турция, Испания, Греция, Таиланд, Тунис, Марокко, Израиль, ОАЭ, Куба, Индия, Маврикий, Танзания, Доминиканская Республика, Индонезия, Мальдивы, Вьетнам, Сейшелы, Шри-Ланка, Сингапур, Мексика, Камбоджа, Иордания, Андорра, Австрия, Азербайджан, Грузия. Идет постоянная работа по открытию новых направлений. Туроператор организует групповые и индивидуальные туры FIT на базе собственных чартерных программ и регулярных рейсов, занимается развитием incentive-, congress-, спортивный и другие виды туризма.

Пользователям также предлагаются готовые туры, которые они могут приобрести в короткое время. Помимо этого, на сервисе присутствуют предложения «горячих» туров.

Аналогично агентству АталанТур, CORAL TRAVEL предоставляет пользователям возможность подбора собственного тура, настраиваемого лично пользователем. Пример интерфейса данного сервиса представлен на рисунке 1.2.

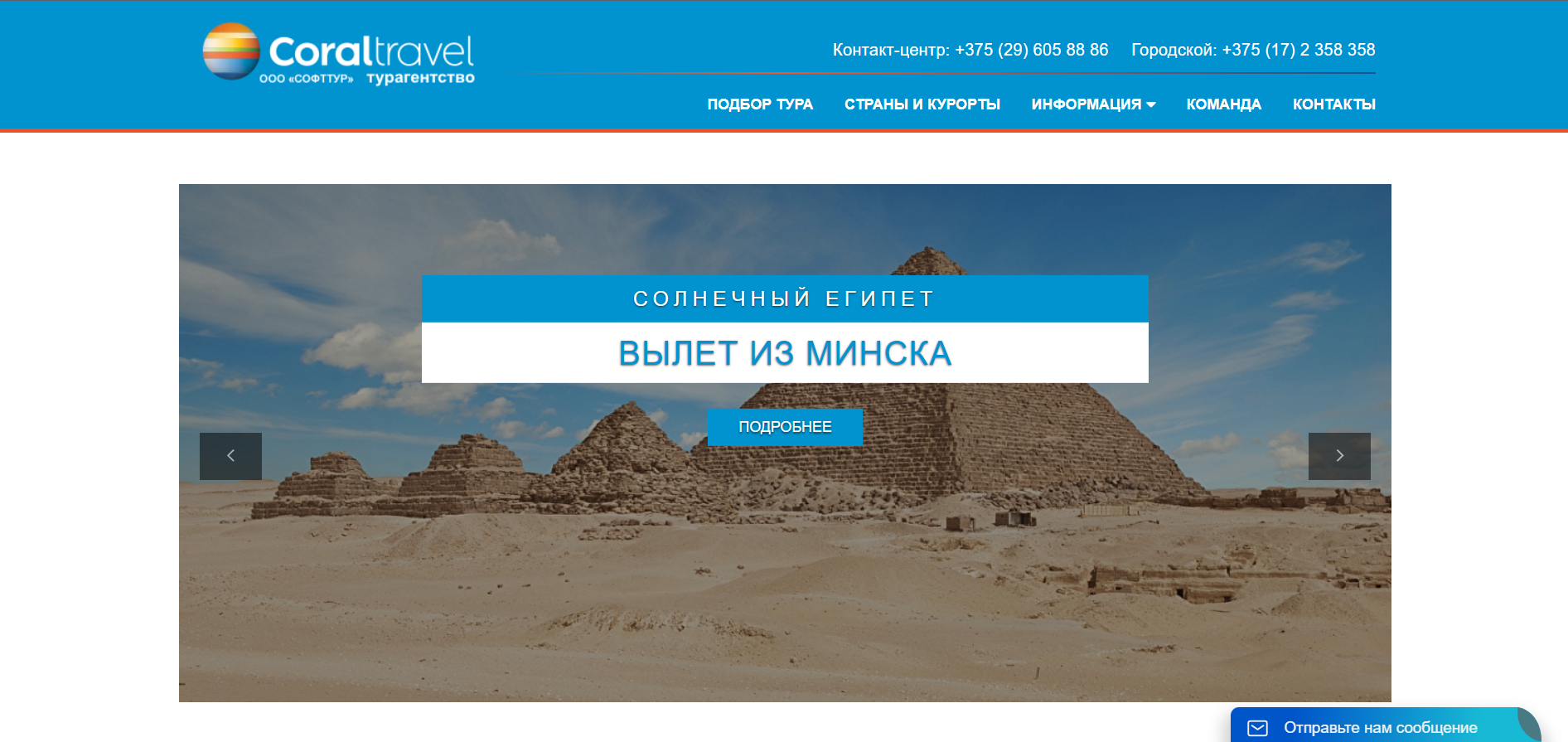


Рисунок 1.2 – Интерфейс сервиса CORAL TRAVEL

В данном разделе были проведены аналитические обзоры двух туристических агентств - АталанТур и CORAL TRAVEL. Оба сервиса предоставляют пользователям возможность приобретения готовых туров или создания своего собственного с необходимыми параметрами.

* 1. Разработка функциональных требований, определение вариантов использования

Функциональные требования описывают функции и сервисы, которые должны быть предоставлены базой данных, такие как ввод, хранение, обработка, анализ и вывод информации. Они определяют основные операции, которые должны быть выполнены в базе данных, и детализируют требования к вводу данных, обработке запросов, управлению транзакциями и т.д. Это может включать в себя описание того, как данные должны храниться и организовываться, как происходит поиск и выборка данных, каким образом обновляются данные и какие механизмы используются для защиты данных. Функциональные требования должны быть конкретными, измеримыми и ясно определенными, чтобы они могли быть реализованы в БД. Они должны также быть документированы и включены в спецификацию требований, которая используется в процессе разработки БД. Это помогает обеспечить соответствие разработанной БД бизнес-потребностям и обеспечивает качество конечного продукта.

Помимо функциональных требований, важно также определить роли пользователей и их варианты использования системы. Каждая роль определяет набор привилегий, которые могут быть предоставлены пользователям, связанным с этой ролью. Например, роль "администратор" может иметь полный доступ ко всем объектам базы данных, включая таблицы, представления, хранимые процедуры и триггеры. Роль "менеджер" может иметь только доступ к определенным таблицам и представлениям, а также права на выполнение определенных запросов. Варианты использования обычно представляются в виде UML диаграмм, которые позволяют наглядно отобразить взаимодействие между пользователями и системой.

UML-диаграмма — это графическое представление модели объектно-ориентированной системы, используемое для описания ее структуры, поведения и взаимодействия между ее компонентами. UML-диаграммы могут включать в себя различные элементы, такие как классы, объекты, интерфейсы, отношения между объектами, состояния и переходы, взаимодействия между объектами и т.д.

Роли пользователя в БД – это механизм управления доступом к данным и объектам в базе данных, который определяет, какие действия могут выполнять пользователи в зависимости от их роли или прав. Роли могут быть назначены для группы пользователей, чтобы упростить управление доступом к базе данных и защитить данные от несанкционированного доступа. В данном проекте роли пользователей будут следующими:

* User.
* Manager.
* Administrator.

На основе предоставленного списка ролей необходимо построить варианты использования в виде UML-диаграммы. Варианты использования изображены на рисунке 1.3.

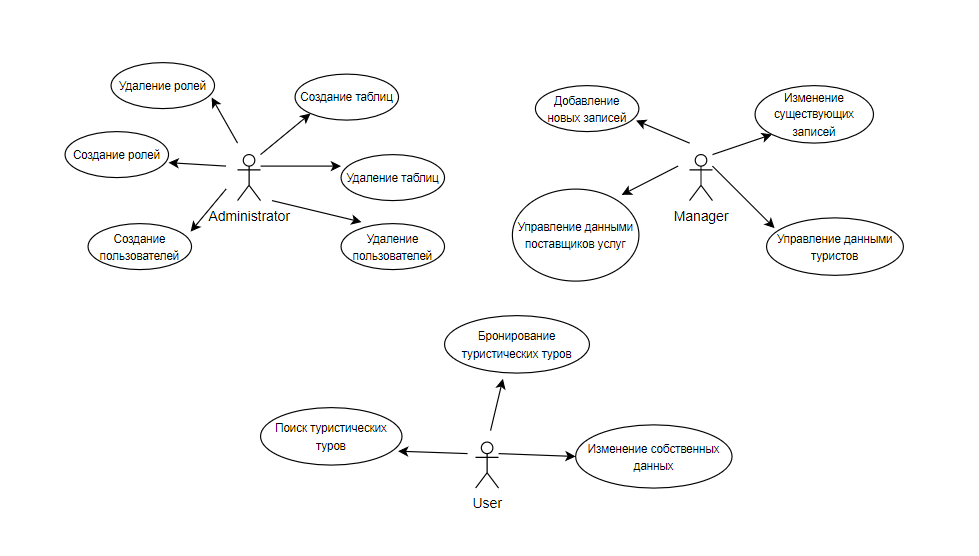


Рисунок 1.3 – UML диаграмма вариантов использования

Роль User получает возможность поиска туристических туров на основе параметров, которые будут предложены ему. Помимо этого, пользователю будет предложена возможность бронирования туров, размещенных на платформе.

Роль Manager заключается в управлении данными о туристических пакетах, предложениях и бронированиях, а также данными о клиентах, сотрудниках и поставщиках услуг. Администратор имеет доступ к просмотру информации о всех пользователях, в том числе их действиях на площадке. Кроме того, менеджер туристического агентства может создавать новые записи в базе данных о туристических пакетах, определять продолжительность, описание и другие характеристики, которые могут быть полезны для пользователей. Он также может управлять данными о поставщиках услуг, таких как авиалинии, гостиницы, рестораны и т.д., и иметь доступ к информации о состоянии их услуг.

Роль Administrator предоставляет полный доступ ко всем объектам базы данных и может выполнять операции по управлению базой данных, такие как создание и удаление таблиц, пользователей и ролей.

В заключении можно отметить, что роли пользователей в базе данных (БД) являются важным аспектом ее проектирования и разработки. Различные пользователи могут иметь различные уровни доступа и функциональные возможности, которые соответствуют их ролям в организации и бизнес-процессам.

Например, менеджеры туристического агентства могут иметь доступ к более широкому диапазону функциональных возможностей, чем обычные клиенты, чтобы иметь возможность управлять продажами, бронированиями и другими аспектами бизнес-процессов. Клиенты могут иметь доступ только к функциям, необходимым для поиска, просмотра и бронирования туров.

1.3 Вывод

В данном разделе был проведен аналитический обзор аналогов туристических платформ и сервисов, которые уже существуют долгое время в сфере туризма. С помощью данного обзора были выделены основные цели и функциональные возможности, которые необходимо реализовать в итоговом продукте. Дополнительно были определены основные реализуемые требования базы данных, а также роли пользователей и варианты использования системы в зависимости от этих ролей. Была разработана UML-диаграмма, на которой отображены основные функции, которые доступны для каждой из ролей пользователей.

1. Разработка архитектуры проекта
   1. Обобщенная структура управлением приложения

Для обеспечения управления приложением с использованием базы данных необходимо разработать удобный и интуитивно понятный интерфейс, который позволит пользователю взаимодействовать с базой данных и эффективно управлять данными. Это может включать в себя разработку оптимизированных запросов для вставки, обновления и удаления данных, а также разработку механизмов для извлечения и обработки информации из базы данных.

В функциональность приложения туристического агентства может входить хранение информации о различных отелях и курортах, их категории, ценах, свободных номерах. Также пользователи должны иметь возможность быстрого поиска и бронирования туров для с учетом их предпочтений и бюджета, формирование документов для оплаты и подтверждения бронирования.

* 1. Диаграммы UML, взаимосвязь всех компонентов.

Диаграмма базы данных таблиц (Database Table Diagram) ­– это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных будет представлена на рисунке 2.1.

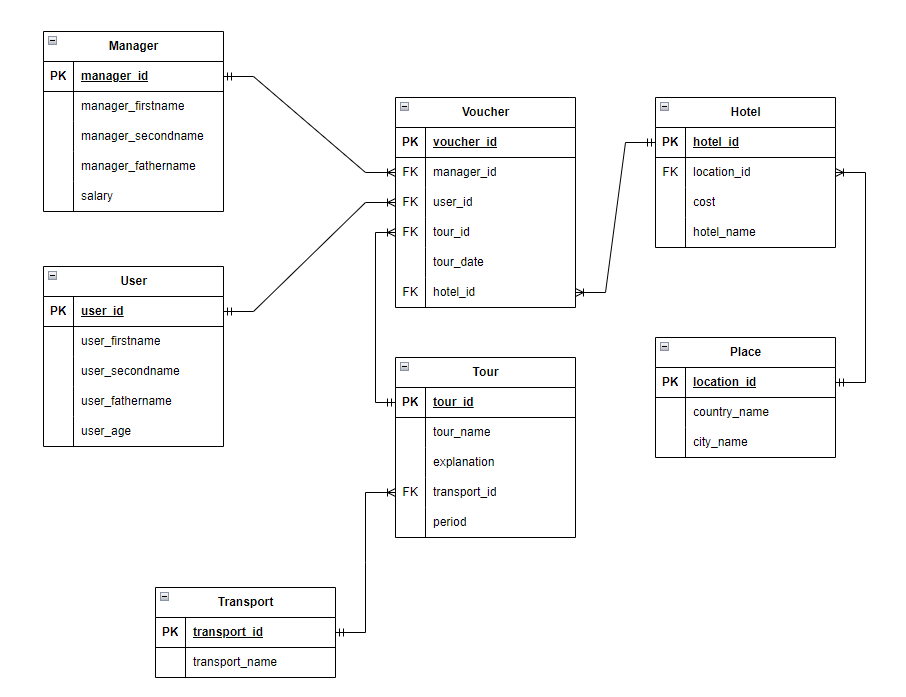


Рисунок 2.1 – Диаграмма базы данных

Таким образом, диаграмма показывает связи между таблицами и полями, а также отношения между ними, такие как связи "один-ко-многим". К примеру, к таблице Voucher связываются еще три таблицы: Manager, User, Tour.

* 1. Описание информационных объектов

Для реализации базы данных было разработано 7 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: Voucher, User, Manager, Hotel, Location, Tour, Transport. Ниже приведен подробный разбор каждой из них.

Таблица Voucher является основной сущностью информационной системы, хранящая информацию о распределении туристов по отелям и заездам. Имеющиеся поля: voucher\_id (идентификатор путевки, типа NUMBER(12), первичный ключ), manager\_id (идентификатор менеджера, тип NUMBER(12), внешний ключ ссылающийся на manager \_id таблицы Manager), user\_id (идентификатор пользователя, тип NUMBER(12), внешний ключ ссылающийся на user\_id таблицы User), tour\_id (идентификатор тура, тип NUMBER(12), внешний ключ ссылающийся на tour\_id таблицы Tour), tour\_date (дата вылета на отдых, тип date), hotel\_id (идентификатор отеля, тип NUMBER(12), внешний ключ ссылающийся на hotel\_id таблицы Hotel).

Таблица Manager содержит информацию о менеджерах. Имеющиеся поля: manager\_id (идентификатор пользователя, типа NUMBER(12), первичный ключ), manager\_firstname (имя менеджера, тип VARCHAR(100)), manager\_secondname (фамилия менеджера, тип VARCHAR(100)), manager\_fathername (отчество менеджера, тип VARCHAR(100)), salary (зарплата менеджера, тип binary\_float).

Таблица User содержит информацию о пользователях. Имеющиеся поля: user\_id (идентификатор туриста, типа NUMBER(12), первичный ключ), user\_firstname (имя туриста, тип VARCHAR(100)), user\_secondname (фамилия туриста, тип VARCHAR(100)), user\_fathername (отчество туриста, тип VARCHAR(100)), user\_age (возраст туриста, тип NUMBER(12)).

Таблица Transport содержит информацию о транспорте, которыми туристы доставляются к отелям. Имеющиеся поля: transport\_id (идентификатор транспорта, тип NUMBER(12), первичный ключ), transport\_name (название транспорта, тип VARCHAR(100)).

Таблица Tour хранит перечень туров в отелях, с указанием продолжительности заезда. Имеющиеся поля: tour\_id (идентификатор тура, типа INTEGER, первичный ключ), tour\_name (название тура, тип VARCHAR(100)), explanation (описание тура, тип VARCHAR(100)), transport\_id (идентификатор транспорта, тип NUMBER(12), внешний ключ ссылающийся на transport\_id таблицы Transport), period (продолжительность, тип NUMBER(12)).

Таблица Place содержит информацию о местоположении отеля. Имеющиеся поля: location\_id (идентификатор метоположения, типа NUMBER(12), первичный ключ), country\_name (название страны, тип VARCHAR(100)), city\_name (название города, тип VARCHAR(100)).

Таблица Hotel содержит информацию о отелях. Имеющиеся поля: hotel\_id (идентификатор отеля, тип NUMBER(12), первичный ключ), location\_id (идентификикатор местоположения, тип NUMBER(12), внешний ключ ссылающийся на location\_id в таблице Location), cost (цена за одну ночь, тип binary\_float), hotel\_name (название отеля, тип VARCHAR(100)).

* 1. Вывод

Разработка архитектуры проекта является важным этапом в создании приложения для туристического агентства. Архитектура проекта определяет структуру и организацию компонентов приложения, а также обеспечивает масштабируемость, гибкость и устойчивость к изменениям.

Этап разработки архитектуры проекта включает в себя проектирование функциональности системы, определение архитектурных стилей, выбор технологий и платформ, разработку диаграмм и моделей, и многое другое. Важно учитывать как функциональные, так и нефункциональные требования, такие как производительность, масштабируемость, безопасность и т.д.

Хорошо спроектированная архитектура может ускорить разработку и снизить затраты на поддержку приложения в долгосрочной перспективе. При разработке архитектуры проекта необходимо учитывать требования бизнеса и конечных пользователей, а также применять лучшие практики и шаблоны проектирования для создания качественного и эффективного приложения.

1. Разработка модели базы данных
   1. Создание необходимых объектов

Для туристического необходимо создать некоторое число таблиц, которые будут хранить данные о пользователях, местоположении, турах, транспорте и т.д.

Для туристического агентства необходимо создать следующие таблицы:

Voucher – основная таблица информационной системы, хранящая информацию о распределении туристов по отелям и заездам. Важными атрибутами этой сущности являются: дата вылета на отдых, тур, отель, турист.

Tour – таблица, отвечающая за хранение перечня туров в отелях, с указанием продолжительности заезда. Важными атрибутами этой сущности являются: название тура, продолжительность, описание, вид транспорта для доставки туристов в отель.

Hotel – таблица, перечень отелей, которые принимают туристов на отдых. В этой таблице будут храниться название отеля, местоположение, цена за сутки и т.д.

Users – таблица, содержащая информацию о туристах, которые совершили туристические туры. В этой таблице будут храниться данные о туристе, такие как имя, возраст и уникальный идентификатор.

Managers – таблица, содержащая информацию о менеджерах фирмы. В этой таблице будут храниться их имена и заработная плата.

Place – таблица, содержащая информацию о местоположении отеля, в который заселяется пользователь. В этой таблице будут храниться данные о городе и стране размещения отеля.

Transport – отвечает за хранение перечня видов транспорта, которым туристы доставляются к отелям. Важным атрибутом этой сущности является название транспорта.

Для более удобной работы с базой данных можно создать несколько функций и процедур. Их использование в базе данных имеет несколько преимуществ. Во-первых, это позволяет уменьшить количество повторяющегося кода, что упрощает сопровождение и обновление базы данных. Во-вторых, процедуры и функции можно использовать для обеспечения безопасности данных, например, чтобы предотвратить неправильный ввод данных. Также они могут быть использованы для оптимизации производительности, позволяя выполнять множество операций в одной транзакции вместо выполнения множества запросов по отдельности. Кроме того, процедуры и функции могут быть использованы для автоматизации определенных процессов в базе данных, что также может повысить производительность и снизить вероятность ошибок.

* + 1. Представления базы данных

Представление (view) в базе данных представляет собой виртуальную таблицу, которая создается на основе запроса к одной или нескольким таблицам в базе данных. В целом, представления упрощают работу с данными в базе данных и позволяют использовать ее более эффективно и безопасно. Они могут также быть использованы для улучшения производительности запросов и минимизации повторного использования кода, так как одно и то же представление может быть использовано в нескольких запросах.

В данном проекте были созданы три представления:

* cost\_hotel\_info, которое содержит информацию о ценах отелей в разных странах и городах, а также упорядочивает их от дорогих к дешевым;
* tours\_for\_any\_weeks, которое содержит информацию о турах, которые запланированы на несколько недель;
* voucher\_info, которое содержит полную информацию о путевке;

Представление cost\_hotel\_info было создано для того, чтобы пользователи могли удобно получить информацию о ценах отелей в определенном месте.

|  |
| --- |
| CREATE VIEW COST\_HOTEL\_INFO AS  SELECT loc.country\_name, loc.city\_name, hot.hotel\_name, hot.cost\_for\_day  FROM Hotel hot, Place loc WHERE hot.location\_id = loc.location\_id  ORDER BY hot.cost\_for\_day DESC; |

Листинг 3.1 – Представление COST\_HOTEL\_INFO

Представление tours\_for\_any\_weeks было создано для того, чтобы получить информацию о турах, которые пройдут в следующем месяце.

|  |
| --- |
| CREATE VIEW TOURS\_FOR\_ANY\_WEEKS AS  SELECT \* FROM Tours  WHERE period > 7; |

Листинг 3.2 – Представление TOURS\_FOR\_ANY\_WEEKS

Представление voucher\_info было создано для того, чтобы получить полную информацию о путевке.

* + 1. Индексы базы данных

Индекс в базе данных (БД) – это структура данных, используемая для ускорения поиска и обновления данных в таблицах. Он содержит ссылки на строки таблицы, отсортированные по значению столбца или группе столбцов. Это позволяет СУБД быстрее находить нужные данные при выполнении запросов. Создание индексов может ускорить выполнение запросов и повысить производительность БД. Однако, индексы могут занимать дополнительное место в БД и замедлять операции обновления и вставки данных. Поэтому выбор индексов должен быть осуществлен с учетом особенностей конкретной БД и запросов, которые будут выполняться.

Для ускорения поиска данных, в работе было сделано несколько индексов:

* FK\_PLACE в таблице Hotel по полю location\_id для быстрого связывания таблиц;
* FK\_TRANSPORT в таблице Tour по полю transport\_id для быстрого связывания таблиц.
* FK\_MANAGERS в таблице Vouchers по полю manager\_id для быстрого связывания таблиц.

|  |
| --- |
| CREATE INDEX FK\_PLACE ON HOTEL (location\_id);  ------TOUR--------  CREATE INDEX FK\_TRANSPORT ON TOUR (transport\_id);  ------VOUCHER-----  CREATE INDEX FK\_MANAGERS ON VOUCHER (manager\_id);  CREATE INDEX FK\_USERS ON VOUCHER (user\_id);  CREATE INDEX FK\_TOUR ON VOUCHER (tour\_id);  CREATE INDEX FK\_HOTEL ON VOUCHER (hotel\_id); |

Листинг 3.3 – Индексы базы данных

* + 1. Триггеры базы данных

Триггер в базе данных – это специальный тип хранимых процедур, которые запускаются автоматически при определенных действиях с данными в таблице. Триггеры могут использоваться для обеспечения целостности данных, автоматизации процессов, контроля изменений и многих других задач.

Триггеры могут быть определены на разных уровнях базы данных - на уровне таблицы, на уровне базы данных или на уровне сервера. Они могут быть вызваны при различных операциях с данными, таких как INSERT, UPDATE и DELETE, а также при изменении схемы базы данных.

Триггеры могут содержать SQL-код, который выполняется при срабатывании триггера, а также ссылки на другие объекты базы данных, такие как таблицы или представления.

Триггер, созданный для таблицы Manager, увеличивает зарплату менеджера (поле salary). Он увеличивает зарплату на 10 процентов в том случае, когда добавляется новая запись в таблицу Voucher с указанием идентификатора конкретного менеджера. Скрипт триггера будет представлен на листинге 3.2.

|  |
| --- |
| CREATE OR REPLACE TRIGGER update\_manager\_salary  AFTER INSERT OR UPDATE OF manager\_id ON Voucher  FOR EACH ROW  BEGIN  UPDATE Manager SET salary = salary \* 1.1 WHERE id = :new.manager\_id;  END; |

Листинг 3.4 – Скрип триггера update\_manager\_salary

Таким образом, данный триггер выполняет увеличение зарплаты менеджера в случае выполнения заданного условия.

Триггер CHECK\_USER\_AGE используется при добавлении или изменении таблицы Users. Его задачей является проверка вводимых данных на соответствие возрасту, который должен быть более 10 лет.

* 1. Описание используемой технологии

Технология мониторинга состояния СУБД является неотъемлемой частью процесса управления базами данных. С помощью мониторинга можно отслеживать работу СУБД и выявлять проблемы до того, как они приведут к серьезным последствиям.

Существуют различные средства мониторинга состояния СУБД, которые позволяют собирать информацию о различных аспектах ее работы, таких как загрузка ЦПУ и памяти, производительность выполнения запросов, доступность сети и т.д.

Одним из наиболее популярных средств мониторинга является Oracle Enterprise Manager (OEM), который позволяет мониторить состояние не только баз данных Oracle, но и других систем, связанных с базами данных. С помощью этого средства можно отслеживать производительность, доступность и надежность баз данных, а также настраивать автоматические оповещения в случае возникновения проблем.

В SQL есть набор системных таблиц и представлений, которые могут быть использованы для мониторинга состояния СУБД. Например, для мониторинга текущей активности в базе данных можно использовать представление V$SESSION. Пример использования приведен на листинге 3.3.

|  |
| --- |
| SELECT sid, serial#, username, status, machine, program  FROM V$SESSION  WHERE status='ACTIVE'; |

Листинг 3.3 – Представление V$SESSION

Этот запрос вернет список всех сессий, которые в настоящее время находятся в активном состоянии, включая имена пользователей, статусы и названия программ, используемых для подключения к базе данных.

Также можно использовать представление V$SQL для мониторинга выполнения SQL-запросов в базе данных. Пример показан на листинге 3.4.

|  |
| --- |
| song.track\_image = `http://localhost:3001/images/${song?.track\_image?.toString("utf-8")}`; |

Листинг 3.4 – Представление V$SQL

Этот запрос вернет список всех SQL-запросов, выполненных в базе данных, включая количество выполнений, общее время выполнения и количество операций ввода/вывода буфера.

* 1. Вывод

В данном разделе была рассмотрена разработка объектов базы данных для туристического агентства. Tакже была описана использованная технология мониторинга состояния СУБД. Важно отметить, что технология мониторинга состояния СУБД должна использоваться регулярно, чтобы обнаружить проблемы в работе баз данных как можно раньше.

1. Установка, настройка и использование Oracle 12c
   1. Установка Oracle 12c

Oracle – это система управления базами данных (СУБД) компании Oracle Corporation. Она является одной из наиболее популярных СУБД в мире и широко используется в корпоративной среде. Oracle предлагает множество функций и инструментов для управления данными, а также обеспечивает высокую производительность, масштабируемость и надежность.

После установки Oracle на сервер, была произведена конфигурация сервера для оптимальной работы с базой данных.

* 1. Создание таблиц

В данном разделе создаются таблицы для базы данных. Но первоначально необходимо создать табличные пространства, в которых будут храниться таблицы.

Табличное пространство – это механизм, который помогает связать объекты базы данных, такие как таблицы, индексы и представления, с файловой системой. Оно позволяет логически разделять объекты базы данных на разные физические устройства или диски, что может улучшить производительность работы с базой данных. Скрипт для создания табличных пространств будет представлен на листинге 4.1.

|  |
| --- |
| CREATE TABLESPACE TS\_TABLES\_KP  DATAFILE 'C:\app\orcl\_wald\tablespaces\TS\_TABLES\_KP.dbf'  SIZE 500M  AUTOEXTEND ON  EXTENT MANAGEMENT LOCAL;  CREATE TABLESPACE TS\_MAIN\_KP  DATAFILE 'C:\app\orcl\_wald\tablespaces\TS\_MAIN\_KP.dbf'  SIZE 500M  AUTOEXTEND ON  EXTENT MANAGEMENT LOCAL;  CREATE TEMPORARY TABLESPACE TS\_TEMP\_KP  TEMPFILE 'C:\app\orcl\_wald\tablespaces\TS\_TEMP\_KP.dbf'  SIZE 100M  AUTOEXTEND ON  EXTENT MANAGEMENT LOCAL; |

Листинг 4.1 – Cкрипт для создания табличных пространств

Для базы данных создадим семь основных таблиц: Voucher, Tour, Transport, Hotel, Place, Manager, User.

Все таблицы будут храниться в одном табличном пространстве — TS\_TABLES\_KP. Это обусловлено тем, что создание отдельного табличного пространства для каждой таблицы приведет к избыточности и усложнению управления базой данных, а также снижению производительности из-за необходимости обращения к разным табличным пространствам при выполнении запросов.

Каждая таблица будет содержать свои поля (столбцы) и ограничения (constraints), которые определяют правила для хранения и изменения данных. Например, ограничение FOREIGN KEY определяет связь между двумя таблицами, а ограничение PRIMARY KEY определяет уникальный идентификатор для каждой записи в таблице.

Кроме того, в базе данных будут присутствовать связи между таблицами. Одна из основных связей — это связь "один ко многим" (one-to-many), которая определяет отношение одной записи в таблице к нескольким записям в другой таблице. Например, у каждого пользователя может быть несколько туров. Для этого мы добавим в таблицу Voucher внешний ключ (FOREIGN KEY) на таблицу User, который будет указывать на идентификатор пользователя.

Скрипт создание таблицы Voucher будет представлен на листинге 4.2.

|  |
| --- |
| create table Voucher (  voucher\_id number(12) not null,  manager\_id number(12) not null,  user\_id number(12) not null,  tour\_id number(12) not null,  tour\_date date,  hotel\_id number(12) not null,  constraint PK\_Voucher primary key(voucher\_id) using index enable,  constraint FK\_MANAGERS foreign key (manager\_id) references Managers (manager\_id),  constraint FK\_USERS foreign key (user\_id) references Users (user\_id),  constraint FK\_TOUR foreign key (tour\_id) references Tour (tour\_id),  constraint FK\_HOTEL foreign key (hotel\_id) references Hotel (hotel\_id)  ) tablespace TS\_TABLES\_KP; |

Листинг 4.2 – Cкрипт создание таблицы Voucher

Таким образом, было описано создание табличного пространства для базы данных, а также таблиц, которые будут храниться в этих пространствах. Были созданы семь табличных пространств, в каждом из которых будут храниться соответствующие таблицы.

* 1. Создание ролей для разграничения

В этом разделе создаются роли для ограничения доступа к базе данных. Создание ролей позволяет установить границы доступа к различным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации.

Будут созданы три роли для разграничения доступа к базе данных: RL\_USER\_KP, RL\_MANAGER\_KP и RL\_ADMIN\_KP. Это позволит ограничить доступ к определенным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации. Роли USER\_KP и RL\_MANAGER\_KP являются основными, в то время как роль RL\_ADMIN\_KP будет иметь более широкий набор привилегий.

Роль RL\_ADMIN\_KP имеет полный доступ ко всей базе данных и может выполнять любые операции, в том числе создавать и изменять таблицы, индексы, представления и триггеры. Выданные привилегии роли programmer\_role можно увидеть на листинге 4.3.

|  |
| --- |
| CREATE ROLE RL\_ADMIN\_KP;  GRANT ALL PRIVILEGES TO RL\_ADMIN\_KP;  CREATE PROFILE PF\_ADMIN\_KP LIMIT  PASSWORD\_LIFE\_TIME 180  SESSIONS\_PER\_USER 3  FAILED\_LOGIN\_ATTEMPTS 7  PASSWORD\_LOCK\_TIME 1  PASSWORD\_REUSE\_TIME 10  PASSWORD\_GRACE\_TIME DEFAULT  CONNECT\_TIME 180  IDLE\_TIME 30;  CREATE USER ADMIN\_KP IDENTIFIED BY 75980  DEFAULT TABLESPACE TS\_ADMIN\_KP QUOTA UNLIMITED ON TS\_ADMIN\_KP  PROFILE PF\_ADMIN\_KP  ACCOUNT UNLOCK;  GRANT RL\_ADMIN\_KP TO ADMIN\_KP; |

Листинг 4.3 – Привилегии, выданные роли RL\_ADMIN\_KP

Роль RL\_MANAGER\_KP имеет права на выполнение различных функций и процедур, связанных с управлением пользователями и треками в базе данных. Эта роль может выполнять функции добавления, редактирования и удаления записей, а также данных об отелях, транспорте и прочем. Выданные привилегии роли MANAGER\_KP можно увидеть на листинге 4.4.

|  |
| --- |
| CREATE ROLE RL\_MANAGER\_KP;  GRANT CREATE SESSION TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON Voucher TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON Tour TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON Transport TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON Hotel TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON Location TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON Manager TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON User TO RL\_MANAGER\_KP; |

Листинг 4.4 – Привилегии, выданные роли RL\_MANAGER\_KP

Роль RL\_USER\_KP может искать туры по необходимым параметрам, просматривать их все, имеет доступ к данным о отелях, транспорте и местоположении отелей.

|  |
| --- |
| CREATE ROLE RL\_USER\_KP;  GRANT CREATE SESSION TO RL\_USER\_KP;  GRANT EXECUTE ON PROCEDURE UPDATE\_USER TO RL\_USER\_KP;  GRANT EXECUTE ON PROCEDURE UPDATE\_USER\_PASSWORD TO RL\_USER\_KP;  GRANT EXECUTE ON FUNCTION GET\_TOUR\_BY\_TIME TO RL\_USER\_KP;  GRANT EXECUTE ON FUNCTION GET\_LOC TO RL\_USER\_KP; |

Листинг 4.5 – Привилегии, выданные роли RL\_USER\_KP

Таким образом, были созданы три роли для ограничения доступа к базе данных: RL\_USER\_KP, RL\_MANAGER\_KP и RL\_ADMIN\_KP. Каждая роль имеет определенный набор привилегий, который позволяет пользователю выполнять определенные функции в базе данных. Роль RL\_ADMIN\_KP имеет наибольшие привилегии и может выполнять любые операции в базе данных, в то время как роль RL\_USER\_KP имеет ограниченный набор привилегий, который позволяет пользователю только просматривать информацию. Роль RL\_MANAGER\_KP имеет средний уровень привилегий и может выполнять функции, связанные с управлением пользователями и таблицами в базе данных.

* 1. Создание пакетов процедур для базы данных

Для управления данными через приложение пользователи и администраторы используют хранимые процедуры и функции. Хранимая процедура представляет собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Функция также представляет собой набор SQL-инструкций, но возвращает значение, которое может быть использовано внутри другой инструкции SQL.

Отличие функций от процедур состоит в том, что функции возвращают значение, которое может быть использовано в других SQL-запросах, а процедуры не возвращают значение. Кроме того, функции могут быть использованы в выражениях SQL, например, для вычисления значения поля в запросе SELECT.

В зависимости от того, какую задачу необходимо выполнить, следует использовать хранимую процедуру или функцию. Хранимые процедуры могут использоваться для выполнения сложных операций над данными, таких как массовые изменения в таблицах, а также для оптимизации производительности приложения. Функции же наиболее полезны в случаях, когда требуется выполнить вычисление на основе данных в базе данных, например, для подсчета статистики или фильтрации данных.

Так как множество процедур и функций являются похожими между собой, в последующих подпунктах будут рассмотрены некоторые них.

* + 1. Выборка данных из таблиц

Для вывода данных из таблиц были написаны следующие процедуры и функции: ALL\_VOUCHER, ALL\_USERS, ALL\_TOUR, ALL\_HOTELи т.д. Основная их задача – выборка данных из всех основных таблиц базы данных. Ниже будут описание каждой функции.

На листинге 4.6 будет процедура ALL\_VOUCHER, которая предназначена для выборки всех туров из таблицы Voucher.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE ALL\_VOUCHER  AS  BEGIN  DECLARE  a\_voucher\_id Voucher.voucher\_id%TYPE;  a\_manager\_id Voucher.manager\_id%TYPE;  a\_user\_id Voucher.user\_id%TYPE;  a\_tour\_id Voucher.tour\_id%TYPE;  a\_tour\_date Voucher.tour\_date%TYPE;  a\_hotel\_id Voucher.hotel\_id%TYPE;  CURSOR c\_vouch IS SELECT \* FROM Voucher;  BEGIN  DBMS\_OUTPUT.ENABLE (buffer\_size => null);  FOR l\_vouch IN c\_vouch  LOOP  a\_voucher\_id := l\_vouch.voucher\_id;  a\_manager\_id := l\_vouch.manager\_id;  a\_user\_id := l\_vouch.user\_id;  a\_tour\_id := l\_vouch.tour\_id;  a\_tour\_date := l\_vouch.tour\_date;  a\_hotel\_id := l\_vouch.hotel\_id;  DBMS\_OUTPUT.PUT\_LINE (a\_voucher\_id ||' | '|| a\_manager\_id ||' | '|| a\_user\_id ||' | '|| a\_tour\_id ||' | '|| a\_tour\_date ||' | '|| a\_hotel\_id);  END LOOP;  END;  EXCEPTION  WHEN OTHERS  THEN  DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END; |

Листинг 4.6 – Процедура ALL\_VOUCHER

Все остальные функции и процедуры будут аналогичны, также предназначены для выборки данных из различных таблиц базы данных.

* + 1. Заполнение таблиц 100 000 строк

Для заполнения таблицы Transport была разработана функция INSERT\_TRANSPORT, которая вставляет 100000 строк в таблицу. Функция представлена на листинге 4.8.

|  |
| --- |
| CREATE OR REPLACE FUNCTION INSERT\_ TRANSPORT()  RETURNS VOID AS  DECLARE  I INT := 1;  BEGIN  WHILE I <= 100000 LOOP  INSERT INTO Transport (transport\_name) VALUES ('Transport ' || I);  I := I + 1;  END LOOP;  END; |

Листинг 4.7 – Функция заполнения таблицы Transport

Функция INSERT\_TRANSPORT была создана для заполнения таблицы Transport 100000 строками. В теле функции используется цикл WHILE, который проходит по значениям от 1 до 100000 и для каждого значения выполняет вставку новой строки в таблицу Transport с именем ' Transport ' и порядковым номером из цикла. Функция не возвращает значение и не принимает аргументов. Для выполнения функции необходимо выполнить SELECT INSERT\_TRANSPORT ().

* + 1. Удаление данных в таблицы

Для удаления данных из базы данных были созданы процедуры: DELETE\_USER, DELETE\_HOTEL, DELETE\_MANAGER и прочие.

На листинге 4.10 будет процедура DELETE\_USER.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_USERS (  d\_user\_id Users.user\_id%TYPE  )  AS  BEGIN  DELETE FROM Users  WHERE Users.user\_id = d\_user\_id;  EXCEPTION  WHEN OTHERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END; |

Листинг 4.8 – Процедура для удаления user

Все остальные процедуры будут аналогичны, также предназначены для удаления соответствующих данных из основных таблиц базы данных.

* + 1. Добавление данных в таблицы

Для добавления данных из базы данных были созданы процедуры: INSERT\_USER, INSERT \_HOTEL, INSERT \_MANAGER и прочие.

На листинге 4.10 представлена процедура INSERT \_HOTEL.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE INSERT\_HOTEL (  location\_id Hotel.location\_id%TYPE,  cost\_for\_day Hotel.cost\_for\_day%TYPE,  hotel\_name Hotel.hotel\_name%TYPE  )  AS  BEGIN  INSERT INTO Hotel(hotel\_id, location\_id, cost\_for\_day, hotel\_name)  VALUES(ID\_HOTEL\_SEQ.NEXTVAL, location\_id, cost\_for\_day, hotel\_name);  END; |

Листинг 4.8 – Процедура для добавления Hotel

* + 1. Экспорт и импорт данных

База данных обладает функциональностью экспорта и импорта данных для таблицы "Users" в формате XML. Это предоставляет возможность переноса данных на другой сервер, резервного копирования данных, а также обеспечивает гибкость в работе с информацией, позволяя использовать данные в различных внешних приложениях.

Для экспорта данных в формате XML была создана процедура export\_to\_xml, которая принимает имя файла. Процедура представлена на листинге 4.9.

|  |
| --- |
| CREATE OR REPLACE FUNCTION export\_to\_xml(p\_file\_name VARCHAR2) RETURN BOOLEAN  IS  l\_ctx DBMS\_XMLGEN.ctxhandle;  l\_xml CLOB;  l\_file UTL\_FILE.file\_type;  BEGIN  -- Инициализация контекста  l\_ctx := DBMS\_XMLGEN.newcontext('SELECT \* FROM Users');    -- Получение XML-строки  l\_xml := DBMS\_XMLGEN.getxml(l\_ctx);    -- Закрытие контекста  DBMS\_XMLGEN.closecontext(l\_ctx);    -- Создание XML-файла и запись данных в него  l\_file := UTL\_FILE.fopen('XML\_DIR', p\_file\_name, 'w', 32767);  UTL\_FILE.put\_line(l\_file, l\_xml);  UTL\_FILE.fclose(l\_file);    RETURN TRUE;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.put\_line('Error: ' || SQLERRM);  RETURN FALSE;  END;  DECLARE  l\_result BOOLEAN;  BEGIN  l\_result := export\_to\_xml('XMLFILE.xml');  IF l\_result THEN  DBMS\_OUTPUT.put\_line('Export successful');  ELSE  DBMS\_OUTPUT.put\_line('Export failed');  END IF;  END; |

Листинг 4.9 – Процедура export\_to\_xml

Для импорта данных из файла в формате XML была создана процедура import\_xml\_file которая принимает имя файла и импортирует данные из файла в таблицу Users. Функция представлена на листинге 4.10.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE import\_xml\_file  AS  l\_xml\_raw RAW(32767);  l\_xml CLOB;  l\_file UTL\_FILE.file\_type;  BEGIN  -- Открытие XML-файла  l\_file := UTL\_FILE.fopen('XML\_DIR', 'XMLFILE.xml', 'r', 32767);  -- Чтение XML-файла в RAW-переменную  UTL\_FILE.get\_raw(l\_file, l\_xml\_raw, DBMS\_LOB.getlength(l\_xml\_raw));  -- Закрытие файла  UTL\_FILE.fclose(l\_file);  -- Преобразование RAW-данных в CLOB  l\_xml := UTL\_RAW.cast\_to\_varchar2(l\_xml\_raw);  -- Вставка данных во временную таблицу  INSERT INTO temp\_users  SELECT \*  FROM XMLTable('/ROWSET/ROW'  PASSING XMLTYPE(l\_xml)  COLUMNS user\_id NUMBER PATH 'USER\_ID',  user\_firstname VARCHAR2(255) PATH 'USER\_FIRSTNAME',  user\_secondname VARCHAR2(255) PATH 'USER\_SECONDNAME',  user\_fathertname VARCHAR2(255) PATH 'USER\_FATHERTNAME',  user\_age NUMBER PATH 'USER\_AGE');  END; |

Листинг 4.10 – Процедура import\_xml

Данные в форматах XML являются универсальным форматом, который может быть использован для передачи данных между различными приложениями и платформами. База данных с возможностью экспорта и импорта данных в форматах XML обеспечивает пользователей удобством и эффективностью управления данными. Таким образом, пользователи могут легко переносить данные между различными серверами или создавать резервные копии данных, используя эту функциональность. Это может значительно сократить время и усилия, затрачиваемые на управление базой данных, и обеспечить более эффективное использование ресурсов.

* 1. Вывод

В разделе создания таблиц, ролей пользователей, процедур и функций базы данных были рассмотрены основные моменты создания и управления структурами данных в Oracle. Были рассмотрены различные типы данных, ключи, а также роли и привилегии пользователей для управления доступом к данным в БД.

Кроме того, были рассмотрены примеры создания процедур и функций для управления данными в БД. Процедуры и функции позволяют ускорить выполнение сложных операций и повторно использовать код в различных частях приложения.

В целом, создание таблиц, ролей пользователей, процедур и функций – это основные элементы, необходимые для создания эффективной и безопасной базы данных в Oracle. При создании структур данных и управлении ими необходимо учитывать требования бизнес-логики и безопасности, а также стремиться к оптимизации производительности и использования ресурсов системы.

1. Тестирование
   1. Тестирование производительности базы данных

Тестирование производительности является одним из ключевых этапов в разработке баз данных, поскольку позволяет оценить ее способность обрабатывать запросы и возвращать результаты в кратчайшие сроки.

В рамках данного проекта было проведено тестирование производительности на таблицах Managers и Voucher, которые связываются с помощью внешнего ключа. Этот выбор был обусловлен необходимостью проверки возможностей базы данных при обработке большого объема информации, что является важным критерием при работе с реальными проектами.

Для получения выборки данных использовался запрос, который представлен на листинге 5.1.

|  |
| --- |
| SET TIMING ON;  SELECT man.manager\_firstname, man.manager\_secondname, vou.user\_id FROM Managers as man INNER JOIN Voucher as vou ON man.manager\_id = vou.manager\_id;  SET TIMING OFF; |

Листинг 5.1 – Запрос к таблицам

Запрос к таблице Polygraphy показывает, что время выполнения запроса составляет 389 мс. Результаты запроса будут представлены на рисунке 5.1.

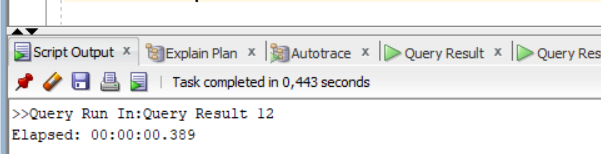


Рисунок 5.1 – Результат выполнения запроса

Для ускорения данного процесса можно создать индекс на поле manager\_id, так как именно по этому полю выполняется фильтрация. После создания индекса, можно повторить запрос и сравнить стоимость с предыдущим запросом. Результат будет представлен на рисунке 5.3



Рисунок 5.2 – Результат выполнения запроса

Данный запрос, после создания индекса показывает, что время выполнения запроса составляет 360 мс, что ускорило запрос на 29мс.

Проведенное тестирование показало, что создание индекса на поле manager\_id привело к заметному улучшению производительности запроса. Время выполнения запроса было сокращено на 29 мс, что при работе с большими объемами данных может значительно сэкономить время. Таким образом, оптимизация запросов путем создания индексов на часто используемых полях может эффективно повысить производительность и ускорить выполнение запросов в базе данных.

* 1. Вывод

В данном разделе была рассмотрена важность тестирования производительности базы данных, а также проведено тестирование на запросе, содержащем в себе связывание двух таблиц. Результаты тестирования показали, что создание индекса на поле manager\_id значительно улучшило производительность запроса, сократив время выполнения на 29 мс. Это свидетельствует о значительном влиянии индексов на производительность базы данных и подчеркивает важность оптимизации запросов для достижения наилучших результатов.

Проведенное тестирование также предоставило ценную информацию о возможностях базы данных в обработке больших объемов данных, что может быть использовано для улучшения ее производительности в целом. Оптимизация запросов и создание индексов на наиболее часто запрашиваемых полях позволяют значительно ускорить выполнение запросов и повысить эффективность работы с базой данных.

1. Руководство использования программного средства
   1. Руководство пользователя

При разработке руководства пользователя требуется создать сценарий, при котором будет протестирована база данных.

Для демонстрации большинства возможностей базы данных требуется создать пользователей: User, Manager, Administrator. После чего продемонстрировать функционал каждого пользователя в базе данных.

После того как User подключается к базе данных, у него появляются возможности разнообразных поисков по параметрам. Они могут включать в себя как процедуры, предоставляющие информацию о таблицах базы данных, так и представления, хранящие в себе все отели, отсортированные по цене. А кроме двух возможностей, у пользователя также есть функционал просмотра различной информации в базе данных. Демонстрация функционала пользователя можно увидеть на листинге 6.1.

|  |
| --- |
| ----User----  EXEC ALL\_VOUCHER;  EXEC ALL\_TOUR;  EXEC ALL\_TRANSPORT;  EXEC ALL\_HOTEL;  EXEC ALL\_PLACE;  EXEC ALL\_MANAGERS;  EXEC ALL\_USERS;  EXEC SELECT\_VOUCHER\_INFO;  EXEC SELECT\_COST\_HOTEL\_INFO;  EXEC SELECT\_TOURS\_FOR\_ANY\_WEEKS; |

Листинг 6.1 – Пример функционала User

Также в базе данных присутствует функционал менеджера, например: добавление нового отеля, изменение стоимости тура, обновление пользовательских данных, а также у менеджера присутствует функционал пользователя. Демонстрация скрипта можно увидеть на листинге 6.2.

|  |
| --- |
| ----Manager----  EXEC ALL\_VOUCHER;  EXEC ALL\_TOUR;  EXEC ALL\_TRANSPORT;  EXEC ALL\_HOTEL;  EXEC ALL\_PLACE;  EXEC ALL\_MANAGERS;  EXEC ALL\_USERS;  EXEC SELECT\_VOUCHER\_INFO;  EXEC SELECT\_COST\_HOTEL\_INFO;  EXEC SELECT\_TOURS\_FOR\_ANY\_WEEKS;  EXEC UPDATE\_USER( '1', 'Andrey’, 'Sik', 'Petrovich', 19);  EXEC INSERT\_USER( '2', 'Maxim’, 'Sik', 'Petrovich', 19); |

Листинг 6.2 – Пример функционала Manager

Дополнительно в базе данных присутствует функционал администратора, который имеет все привилегии работы с данными. Например: удаление таблицы, остановка работы базы данных, удаление строки др.. Демонстрация скрипта можно увидеть на листинге 6.3.

|  |
| --- |
| ----Administrator----  EXEC ALL\_VOUCHER;  EXEC ALL\_TOUR;  EXEC ALL\_TRANSPORT;  EXEC ALL\_HOTEL;  EXEC ALL\_PLACE;  EXEC ALL\_MANAGERS;  EXEC ALL\_USERS;  EXEC SELECT\_VOUCHER\_INFO;  EXEC SELECT\_COST\_HOTEL\_INFO;  EXEC SELECT\_TOURS\_FOR\_ANY\_WEEKS;  EXEC UPDATE\_USER( '1', 'Andrey’, 'Sik', 'Petrovich', 19);  EXEC INSERT\_USER( '2', 'Maxim’, 'Sik', 'Petrovich', 19);  DROP TABLE Voucher;  DELETE FROM Users WHERE user\_id = ‘1’; |

Листинг 6.3 – Пример функционала Administrator

* 1. Вывод

Важно понимать, что база данных является ключевой составляющей любой системы, которая обрабатывает большое количество информации. При разработке такой системы необходимо уделять особое внимание функционалу базы данных, чтобы обеспечить эффективную и безопасную работу пользователей.

Разработчики базы данных должны определить, какие функциональные возможности будут доступны каждому пользователю в зависимости от его роли. Например, пользователь должен иметь доступ только к части информации, в то время как администратор и менеджер должны иметь расширенный функционал.

Помимо функциональных возможностей, необходимо также обеспечить безопасность базы данных и управление доступом к ее содержимому. Для этого важно правильно настроить права доступа к данным, чтобы каждый пользователь имел доступ только к той информации, которая ему нужна. Это поможет избежать утечки данных и несанкционированного доступа.

# Заключение

База данных является важной составляющей информационной системы любой организации. В данном случае была создана база данных для туристического агентства, которая включает в себя таблицы для хранения информации о клиентах, менеджерах, турах, отелях, бронированиях, а также таблицы для хранения информации о ролях пользователей и сотрудников агентства.

Были созданы процедуры и функции, которые обеспечивают управление данными, выборку информации из базы данных, а также взаимодействие между таблицами. Также были созданы роли пользователей, которые позволяют ограничить доступ к определенным данным базы данных в зависимости от уровня доступа пользователя.

Для оптимизации производительности базы данных были добавлены индексы на ключевые поля таблиц. Также были созданы триггеры, которые обеспечивают автоматическое обновление информации при внесении изменений в таблицы.

Тестирование базы данных было проведено при использовании большого объема данных, и результаты оказались положительными. Были разработаны процедуры для импорта и экспорта данных в формате XML, что обеспечило удобство использования и управления данными.

Для обеспечения надежности базы данных были применены средства мониторинга состояния СУБД Oracle, что позволяет оперативно реагировать на возникающие проблемы и устранять их.

Таким образом, создание базы данных туристического агентства позволит организовать эффективную работу агентства, обеспечивая хранение, обработку и защиту информации, а также повышение производительности и надежности базы данных.

# Список литературных источников

1. Habr [Электронный ресурс] / Режим доступа: https://habr.com/ru/all/ – Дата доступа: 22.04.2023.

2. АталанТур [Электронный ресурс] / Режим доступа: https://alatantour.by/ – Дата доступа: 08.04.2023.

3. CoralTravel [Электронный ресурс] / Режим доступа: https://www.coral.by/– Дата доступа: 08.04.2023.

4. Medium «Oracle Programing: Functions vs Procedures» [Электронный ресурс] / Режим доступа: https://medium.com/swlh/oracle-programing-functions-vs-procedures-268108cb49ef – Дата доступа: 20.04.2023.

5. Habr «Все о триггерах» [Электронный ресурс] / Режим доступа: https://habr.com/ru/articles/256655/ – Дата доступа: 25.04.2023.

6. Oracle «6 Data Types» [Электронный ресурс] / Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlqr/Data-Types.html – Дата доступа: 06.05.2023.

# Приложение А

|  |
| --- |
| CREATE TABLESPACE TS\_TABLES\_KP  DATAFILE 'C:\app\orcl\_wald\tablespaces\TS\_TABLES\_KP.dbf'  SIZE 500M  AUTOEXTEND ON  EXTENT MANAGEMENT LOCAL;  CREATE TABLESPACE TS\_MAIN\_KP  DATAFILE 'C:\app\orcl\_wald\tablespaces\TS\_MAIN\_KP.dbf'  SIZE 500M  AUTOEXTEND ON  EXTENT MANAGEMENT LOCAL;  CREATE TEMPORARY TABLESPACE TS\_TEMP\_KP  TEMPFILE 'C:\app\orcl\_wald\tablespaces\TS\_TEMP\_KP.dbf'  SIZE 100M  AUTOEXTEND ON  EXTENT MANAGEMENT LOCAL; |

Листинг 1 – Содержание Tablespace

|  |
| --- |
| create table Voucher (  voucher\_id number(12) not null,  manager\_id number(12) not null,  user\_id number(12) not null,  tour\_id number(12) not null,  tour\_date date,  hotel\_id number(12) not null,  constraint PK\_Voucher primary key(voucher\_id) using index enable,  constraint FK\_MANAGERS foreign key (manager\_id) references Managers (manager\_id),  constraint FK\_USERS foreign key (user\_id) references Users (user\_id),  constraint FK\_TOUR foreign key (tour\_id) references Tour (tour\_id),  constraint FK\_HOTEL foreign key (hotel\_id) references Hotel (hotel\_id)  ) tablespace TS\_TABLES\_KP;  create table Tour (  tour\_id number(12) not null,  tour\_name varchar(100) not null,  explanation varchar(100),  transport\_id number(12) not null,  period number(12),  constraint PK\_Tour primary key(tour\_id) using index enable,  constraint FK\_TRANSPORT foreign key (transport\_id) references Transport (transport\_id)  ) tablespace TS\_TABLES\_KP;  create table Transport (  transport\_id number(12) not null,  transport\_name varchar(100) not null,  constraint PK\_Transport primary key(transport\_id) using index enable  ) tablespace TS\_TABLES\_KP;  create table Hotel (  hotel\_id number(12) not null,  location\_id number(12) not null,  cost\_for\_day binary\_float,  hotel\_name varchar(100) not null,  constraint PK\_Hotel primary key(hotel\_id) using index enable,  constraint FK\_PLACE foreign key (location\_id) references Place (location\_id)  ) tablespace TS\_TABLES\_KP;  create table Place (  location\_id number(12) not null,  country\_name varchar(100) not null,  city\_name varchar(100) not null,  constraint PK\_Place primary key(location\_id) using index enable  ) tablespace TS\_TABLES\_KP;  create table Managers (  manager\_id number(12) not null,  manager\_firstname varchar2(100) not null,  manager\_secondname varchar2(100) not null,  manager\_fathertname varchar2(100),  salary binary\_float,  constraint PK\_Managers primary key(manager\_id) using index enable  ) tablespace TS\_TABLES\_KP;  create table Users (  user\_id number(12) not null,  user\_firstname varchar2(100) not null,  user\_secondname varchar2(100) not null,  user\_fathertname varchar2(100),  user\_age int,  constraint PK\_Users primary key(user\_id) using index enable  ) tablespace TS\_TABLES\_KP; |

Листинг 2 – Создание таблиц

|  |
| --- |
| CREATE ROLE RL\_ADMIN\_KP;  GRANT ALL PRIVILEGES TO RL\_ADMIN\_KP;  CREATE PROFILE PF\_ADMIN\_KP LIMIT  PASSWORD\_LIFE\_TIME 180  SESSIONS\_PER\_USER 3  FAILED\_LOGIN\_ATTEMPTS 7  PASSWORD\_LOCK\_TIME 1  PASSWORD\_REUSE\_TIME 10  PASSWORD\_GRACE\_TIME DEFAULT  CONNECT\_TIME 180  IDLE\_TIME 30;  CREATE USER ADMIN\_KP IDENTIFIED BY 75980  DEFAULT TABLESPACE TS\_MAIN\_KP QUOTA UNLIMITED ON TS\_MAIN\_KP  TEMPORARY TABLESPACE TS\_TEMP\_KP  PROFILE PF\_ADMIN\_KP  ACCOUNT UNLOCK;  GRANT RL\_ADMIN\_KP TO ADMIN\_KP;  drop role RL\_ADMIN\_KP;  drop user ADMIN\_KP;  drop profile PF\_ADMIN\_KP;  drop tablespace TS\_ADMIN\_KP;  CREATE ROLE RL\_MANAGER\_KP;  GRANT CREATE SESSION TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON Voucher TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON Tour TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON Transport TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON Hotel TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON Location TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON Manager TO RL\_MANAGER\_KP;  GRANT INSERT, UPDATE, DELETE ON User TO RL\_MANAGER\_KP;  CREATE PROFILE PF\_MANAGER\_KP LIMIT  PASSWORD\_LIFE\_TIME 180  SESSIONS\_PER\_USER 3  FAILED\_LOGIN\_ATTEMPTS 7  PASSWORD\_LOCK\_TIME 1  PASSWORD\_REUSE\_TIME 10  PASSWORD\_GRACE\_TIME DEFAULT  CONNECT\_TIME 180  IDLE\_TIME 30;  CREATE USER MANAGER\_KP IDENTIFIED BY 75980  DEFAULT TABLESPACE TS\_MAIN\_KP QUOTA UNLIMITED ON TS\_MAIN\_KP  TEMPORARY TABLESPACE TS\_TEMP\_KP  PROFILE PF\_MANAGER\_KP  ACCOUNT UNLOCK;  GRANT RL\_MANAGER\_KP TO MANAGER\_KP;  drop role RL\_MANAGER\_KP;  drop user MANAGER\_KP;  drop profile PF\_MANAGER\_KP;  drop tablespace TS\_MANAGER\_KP;  CREATE ROLE RL\_USER\_KP;  GRANT ALL PRIVILEGES TO RL\_USER\_KP;  GRANT CREATE SESSION TO RL\_USER\_KP;  CREATE PROFILE PF\_USER\_KP LIMIT  PASSWORD\_LIFE\_TIME 180  SESSIONS\_PER\_USER 3  FAILED\_LOGIN\_ATTEMPTS 7  PASSWORD\_LOCK\_TIME 1  PASSWORD\_REUSE\_TIME 10  PASSWORD\_GRACE\_TIME DEFAULT  CONNECT\_TIME 180  IDLE\_TIME 30;  CREATE USER USER\_KP IDENTIFIED BY 75980  DEFAULT TABLESPACE TS\_MAIN\_KP QUOTA UNLIMITED ON TS\_MAIN\_KP  TEMPORARY TABLESPACE TS\_TEMP\_KP  PROFILE PF\_USER\_KP  ACCOUNT UNLOCK;  GRANT RL\_USER\_KP TO USER\_KP;  --GRANT CREATE SESSION TO RL\_ADMIN\_KP;  --GRANT SELECT ON POLICYE TO RL\_ADMIN\_KP;  --GRANT SELECT ON CLIENTE TO RL\_ADMIN\_KP;  --GRANT SELECT ON AGENTE TO RL\_ADMIN\_KP;  --GRANT SELECT ON INSURED\_EVENT TO RL\_ADMIN\_KP;  --GRANT SELECT ON INSURANCE TO RL\_ADMIN\_KP;  --GRANT SELECT ON ORGANIZATION\_INFO TO RL\_ADMIN\_KP;  --GRANT SELECT ON CALCULATIONS TO RL\_ADMIN\_KP;  --GRANT SELECT ON OPERATION TO RL\_ADMIN\_KP;  GRANT SELECT ON ALL\_INSURED\_EVENT TO RL\_ADMIN\_KP;  GRANT SELECT ON ACTIVE\_POLICY TO RL\_ADMIN\_KP;  GRANT SELECT ON INACTIVE\_POLICY TO RL\_ADMIN\_KP;  GRANT SELECT ON LAST\_30DAY\_POLICY TO RL\_ADMIN\_KP;  GRANT SELECT ON LAST\_30DAY\_OPERATIONS TO RL\_ADMIN\_KP;  GRANT SELECT ON LAST\_30DAY\_PAID TO RL\_ADMIN\_KP;  GRANT SELECT ON LAST\_30DAY\_RECEIVED TO RL\_ADMIN\_KP;  GRANT SELECT ON ALL\_YE\_AGENTE TO RL\_ADMIN\_KP;  GRANT SELECT ON ALL\_AGENTE TO RL\_ADMIN\_KP;  GRANT EXECUTE ON SUM\_PAID TO RL\_ RL\_ADMIN\_KP;  GRANT EXECUTE ON SUM\_RECEIVED TO RL\_ADMIN\_KP;  GRANT EXECUTE ON INSERT\_POLICYE TO RL\_ADMIN\_KP;  GRANT EXECUTE ON INSERT\_CLIENTE TO RL\_ADMIN\_KP;  GRANT EXECUTE ON INSERT\_CALCULATIONS TO RL\_ADMIN\_KP;  GRANT EXECUTE ON INSERT\_AGENTE TO RL\_ADMIN\_KP;  GRANT EXECUTE ON INSERT\_INSURANCE TO RL\_ADMIN\_KP;  GRANT EXECUTE ON INSERT\_INSURED\_EVENT TO RL\_ADMIN\_KP;  GRANT EXECUTE ON UPDATE\_POLICYE TO RL\_ADMIN\_KP;  GRANT EXECUTE ON UPDATE\_CLIENTE TO RL\_ADMIN\_KP;  GRANT EXECUTE ON UPDATE\_CALCULATIONS TO RL\_ADMIN\_KP;  GRANT EXECUTE ON UPDATE\_AGENTE TO RL\_ADMIN\_KP;  GRANT EXECUTE ON UPDATE\_INSURANCE TO RL\_ADMIN\_KP;  GRANT EXECUTE ON UPDATE\_INSURED\_EVENT TO RL\_ADMIN\_KP;  GRANT EXECUTE ON UPDATE\_ORGANIZATION\_INFO TO RL\_ADMIN\_KP;  GRANT EXECUTE ON SEARCH\_POLICYE TO RL\_ADMIN\_KP;  GRANT EXECUTE ON SEARCH\_CLIENTE TO RL\_ADMIN\_KP;  GRANT EXECUTE ON SEARCH\_CLIENTE\_FSM TO RL\_ADMIN\_KP;  GRANT EXECUTE ON SEARCH\_AGENTE\_FSM TO RL\_ADMIN\_KP;  GRANT EXECUTE ON SEARCH\_CALCULATIONS\_ID TO RL\_ADMIN\_KP;  GRANT EXECUTE ON SEARCH\_INSURED\_EVENT\_ID TO RL\_ADMIN\_KP;  GRANT EXECUTE ON SEARCH\_INSURED\_EVENT\_NAME TO RL\_ADMIN\_KP;  GRANT EXECUTE ON SEARCH\_AGENTE TO RL\_ADMIN\_KP;  GRANT EXECUTE ON ALL\_POLICYE TO MANAGER\_KP;  GRANT EXECUTE ON ALL\_INSURANCE TO MANAGER\_KP;  GRANT EXECUTE ON ALL\_CLIENTE TO MANAGER\_KP;  GRANT EXECUTE ON ALL\_AGENTE TO MANAGER\_KP;  GRANT EXECUTE ON ALL\_CALCULATIONS TO MANAGER\_KP; |

Листинг 3 – Создание ролей и привилегий

|  |
| --- |
| ---------------------------INDEXES---------------  ------HOTEL-------  CREATE INDEX FK\_PLACE ON HOTEL (location\_id);  ------TOUR--------  CREATE INDEX FK\_TRANSPORT ON TOUR (transport\_id);  ------VOUCHER-----  CREATE INDEX FK\_MANAGERS ON VOUCHER (manager\_id);  CREATE INDEX FK\_USERS ON VOUCHER (user\_id);  CREATE INDEX FK\_TOUR ON VOUCHER (tour\_id);  CREATE INDEX FK\_HOTEL ON VOUCHER (hotel\_id);  DROP INDEX FK\_PLACE;  DROP INDEX FK\_TRANSPORT;  DROP INDEX FK\_MANAGERS;  DROP INDEX FK\_USERS;  DROP INDEX FK\_TOUR;  DROP INDEX FK\_HOTEL;  ------------------------SEQUENCES--------------------  -----VOUCHER  CREATE SEQUENCE ID\_VOUCHER\_SEQ  START WITH 1  INCREMENT BY 1  MAXVALUE 10000000000  NOCYCLE;  -----TOUR  CREATE SEQUENCE ID\_TOUR\_SEQ  START WITH 1  INCREMENT BY 1  MAXVALUE 10000000000  NOCYCLE;  -----TRANSPORT  CREATE SEQUENCE ID\_TRANSPORT\_SEQ  START WITH 1  INCREMENT BY 1  MAXVALUE 10000000000  NOCYCLE;  -----HOTEL  CREATE SEQUENCE ID\_HOTEL\_SEQ  START WITH 1  INCREMENT BY 1  MAXVALUE 10000000000  NOCYCLE;  -----PLACE  CREATE SEQUENCE ID\_PLACE\_SEQ  START WITH 1  INCREMENT BY 1  MAXVALUE 10000000000  NOCYCLE;  -----MANAGERS  CREATE SEQUENCE ID\_MANAGERS\_SEQ  START WITH 1  INCREMENT BY 1  MAXVALUE 10000000000  NOCYCLE;  -----USERS  CREATE SEQUENCE ID\_USERS\_SEQ  START WITH 1  INCREMENT BY 1  MAXVALUE 10000000000  NOCYCLE;  DROP SEQUENCE ID\_VOUCHER\_SEQ;  DROP SEQUENCE ID\_TOUR\_SEQ;  DROP SEQUENCE ID\_TRANSPORT\_SEQ;  DROP SEQUENCE ID\_HOTEL\_SEQ;  DROP SEQUENCE ID\_PLACE\_SEQ;  DROP SEQUENCE ID\_MANAGERS\_SEQ;  DROP SEQUENCE ID\_USERS\_SEQ;  --------------------------------------INSERT-------------------------------  ----VOUCHER---  CREATE OR REPLACE PROCEDURE INSERT\_VOUCHER (  manager\_id Voucher.manager\_id%TYPE,  user\_id Voucher.user\_id%TYPE,  tour\_id Voucher.tour\_id%TYPE,  tour\_date Voucher.tour\_date%TYPE,  hotel\_id Voucher.hotel\_id%TYPE  )  AS  BEGIN  INSERT INTO Voucher(voucher\_id, manager\_id, user\_id, tour\_id, tour\_date, hotel\_id)  VALUES(ID\_VOUCHER\_SEQ.NEXTVAL, manager\_id, user\_id, tour\_id, tour\_date, hotel\_id);  END;  ----TOUR----  CREATE OR REPLACE PROCEDURE INSERT\_TOUR (  tour\_name Tour.tour\_name%TYPE,  explanation Tour.explanation%TYPE,  transport\_id Tour.transport\_id%TYPE,  period Tour.period%TYPE  )  AS  BEGIN  INSERT INTO Tour(tour\_id, tour\_name, explanation, transport\_id, period)  VALUES(ID\_TOUR\_SEQ.NEXTVAL, tour\_name, explanation, transport\_id, period);  END;  ----TRANSPORT----  CREATE OR REPLACE PROCEDURE INSERT\_TRANSPORT (  transport\_name Transport.transport\_name%TYPE  )  AS  BEGIN  INSERT INTO Transport(transport\_id, transport\_name)  VALUES(ID\_TRANSPORT\_SEQ.NEXTVAL, transport\_name);  END;  ----HOTEL----  CREATE OR REPLACE PROCEDURE INSERT\_HOTEL (  location\_id Hotel.location\_id%TYPE,  cost\_for\_day Hotel.cost\_for\_day%TYPE,  hotel\_name Hotel.hotel\_name%TYPE  )  AS  BEGIN  INSERT INTO Hotel(hotel\_id, location\_id, cost\_for\_day, hotel\_name)  VALUES(ID\_HOTEL\_SEQ.NEXTVAL, location\_id, cost\_for\_day, hotel\_name);  END;  ----PLACE----  CREATE OR REPLACE PROCEDURE INSERT\_PLACE (  country\_name Place.country\_name%TYPE,  city\_name Place.city\_name%TYPE  )  AS  BEGIN  INSERT INTO Place(location\_id, country\_name, city\_name)  VALUES(ID\_PLACE\_SEQ.NEXTVAL, country\_name, city\_name);  END;  ----MANAGERS----  CREATE OR REPLACE PROCEDURE INSERT\_MANAGERS (  manager\_firstname Managers.manager\_firstname%TYPE,  manager\_secondname Managers.manager\_secondname%TYPE,  manager\_fathertname Managers.manager\_fathertname%TYPE,  salary Managers.salary%TYPE  )  AS  BEGIN  INSERT INTO Managers(manager\_id, manager\_firstname, manager\_secondname, manager\_fathertname, salary)  VALUES(ID\_MANAGERS\_SEQ.NEXTVAL, manager\_firstname, manager\_secondname, manager\_fathertname, salary);  END;  ----USERS----  CREATE OR REPLACE PROCEDURE INSERT\_USERS (  user\_firstname Users.user\_firstname%TYPE,  user\_secondname Users.user\_secondname%TYPE,  user\_fathertname Users.user\_fathertname%TYPE,  user\_age Users.user\_age%TYPE  )  AS  BEGIN  INSERT INTO Users(user\_id, user\_firstname, user\_secondname, user\_fathertname, user\_age)  VALUES(ID\_USERS\_SEQ.NEXTVAL, user\_firstname, user\_secondname, user\_fathertname, user\_age);  END;  DROP PROCEDURE INSERT\_VOUCHER;  DROP PROCEDURE INSERT\_TOUR;  DROP PROCEDURE INDERT\_TRANSPORT;  DROP PROCEDURE INSERT\_HOTEL;  DROP PROCEDURE INSERT\_PLACE;  DROP PROCEDURE INSERT\_MANAGERS;  DROP PROCEDURE INSERT\_USERS;  -------------------------------------UPDATE-----------------------------  ----VOUCHER---  CREATE OR REPLACE PROCEDURE UPDATE\_VOUCHER (  u\_voucher\_id Voucher.voucher\_id%TYPE,  u\_manager\_id Voucher.manager\_id%TYPE,  u\_user\_id Voucher.user\_id%TYPE,  u\_tour\_id Voucher.tour\_id%TYPE,  u\_tour\_date Voucher.tour\_date%TYPE,  u\_hotel\_id Voucher.hotel\_id%TYPE  )  AS  BEGIN  UPDATE Voucher  SET  Voucher.manager\_id = u\_manager\_id,  Voucher.user\_id = u\_user\_id,  Voucher.tour\_id = u\_tour\_id,  Voucher.tour\_date = u\_tour\_date,  Voucher.hotel\_id = u\_hotel\_id  WHERE  Voucher.voucher\_id = u\_voucher\_id;  EXCEPTION  WHEN OTERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  ----TOUR----  CREATE OR REPLACE PROCEDURE UPDATE\_TOUR (  u\_tour\_id Tour.tour\_id%TYPE,  u\_tour\_name Tour.tour\_name%TYPE,  u\_explanation Tour.explanation%TYPE,  u\_transport\_id Tour.transport\_id%TYPE,  u\_period Tour.period%TYPE  )  AS  BEGIN  UPDATE Tour  SET  Tour.tour\_name = u\_tour\_name,  Tour.explanation = u\_explanation,  Tour.transport\_id = u\_transport\_id,  Tour.period = u\_period  WHERE  Tour.tour\_id = u\_tour\_id;  EXCEPTION  WHEN OTERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  ----TRANSPORT----  CREATE OR REPLACE PROCEDURE UPDATE\_TRANSPORT (  u\_transport\_id Transport.transport\_id%TYPE,  u\_transport\_name Transport.transport\_name%TYPE  )  AS  BEGIN  UPDATE Transport  SET  Transport.transport\_name = u\_transport\_name  WHERE  Transport.transport\_id = u\_transport\_id;  EXCEPTION  WHEN OTERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  ----HOTEL----  CREATE OR REPLACE PROCEDURE UPDATE\_HOTEL (  u\_hotel\_id Hotel.hotel\_id%TYPE,  u\_location\_id Hotel.location\_id%TYPE,  u\_cost\_for\_day Hotel.cost\_for\_day%TYPE,  u\_hotel\_name Hotel.hotel\_name%TYPE  )  AS  BEGIN  UPDATE Hotel  SET  Hotel.location\_id = u\_location\_id,  Hotel.cost\_for\_day = u\_cost\_for\_day,  Hotel.hotel\_name = u\_hotel\_name  WHERE  Hotel.hotel\_id = u\_hotel\_id;  EXCEPTION  WHEN OTERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  ----PLACE----  CREATE OR REPLACE PROCEDURE UPDATE\_PLACE (  u\_location\_id Place.location\_id%TYPE,  u\_country\_name Place.country\_name%TYPE,  u\_city\_name Place.city\_name%TYPE  )  AS  BEGIN  UPDATE Place  SET  Place.country\_name = u\_country\_name,  Place.city\_name = u\_city\_name  WHERE  Place.location\_id = u\_location\_id;  EXCEPTION  WHEN OTERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  ----MANAGERS----  CREATE OR REPLACE PROCEDURE UPDATE\_MANAGERS (  u\_manager\_id Managers.manager\_id%TYPE,  u\_manager\_firstname Managers.manager\_firstname%TYPE,  u\_manager\_secondname Managers.manager\_secondname%TYPE,  u\_manager\_fathertname Managers.manager\_fathertname%TYPE,  u\_salary Managers.salary%TYPE  )  AS  BEGIN  UPDATE Managers  SET  Managers.manager\_firstname = u\_manager\_firstname,  Managers.manager\_secondname = u\_manager\_secondname,  Managers.manager\_fathertname = u\_manager\_fathertname,  Managers.salary = u\_salary  WHERE  Managers.manager\_id = u\_manager\_id;  EXCEPTION  WHEN OTERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  ----USERS----  CREATE OR REPLACE PROCEDURE UPDATE\_USERS (  u\_user\_id Users.user\_id%TYPE,  u\_user\_firstname Users.user\_firstname%TYPE,  u\_user\_secondname Users.user\_secondname%TYPE,  u\_user\_fathertname Users.user\_fathertname%TYPE,  u\_user\_age Users.user\_age%TYPE  )  AS  BEGIN  UPDATE Users  SET  Users.user\_firstnamee = u\_user\_firstname,  Users.user\_secondname = u\_user\_secondname,  Users.user\_fathertname = u\_user\_fathertname,  Users.user\_age = u\_user\_age  WHERE  Users.user\_id = u\_user\_id;  EXCEPTION  WHEN OTERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  DROP PROCEDURE UPDATE\_VOUCHER;  DROP PROCEDURE UPDATE\_TOUR;  DROP PROCEDURE UPDATE\_TRANSPORT;  DROP PROCEDURE UPDATE\_HOTEL;  DROP PROCEDURE UPDATE\_PLACE;  DROP PROCEDURE UPDATE\_MANAGERS;  DROP PROCEDURE UPDATE\_USERS;  ---------------------------ALL------------------------  ----VOUCHER---  CREATE OR REPLACE PROCEDURE ALL\_VOUCHER  AS  BEGIN  DECLARE  a\_voucher\_id Voucher.voucher\_id%TYPE;  a\_manager\_id Voucher.manager\_id%TYPE;  a\_user\_id Voucher.user\_id%TYPE;  a\_tour\_id Voucher.tour\_id%TYPE;  a\_tour\_date Voucher.tour\_date%TYPE;  a\_hotel\_id Voucher.hotel\_id%TYPE;  CURSOR c\_vouch IS SELECT \* FROM Voucher;  BEGIN  DBMS\_OUTPUT.ENABLE (buffer\_size => null);  FOR l\_vouch IN c\_vouch  LOOP  a\_voucher\_id := l\_vouch.voucher\_id;  a\_manager\_id := l\_vouch.manager\_id;  a\_user\_id := l\_vouch.user\_id;  a\_tour\_id := l\_vouch.tour\_id;  a\_tour\_date := l\_vouch.tour\_date;  a\_hotel\_id := l\_vouch.hotel\_id;  DBMS\_OUTPUT.PUT\_LINE (a\_voucher\_id ||' | '|| a\_manager\_id ||' | '|| a\_user\_id ||' | '|| a\_tour\_id ||' | '|| a\_tour\_date ||' | '|| a\_hotel\_id);  END LOOP;  END;  EXCEPTION  WHEN OTHERS  THEN  DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  ----TOUR----  CREATE OR REPLACE PROCEDURE ALL\_TOUR  AS  BEGIN  DECLARE  a\_tour\_id Tour.tour\_id%TYPE;  a\_tour\_name Tour.tour\_name%TYPE;  a\_explanation Tour.explanation%TYPE;  a\_transport\_id Tour.transport\_id%TYPE;  a\_period Tour.period%TYPE;  CURSOR c\_tr IS SELECT \* FROM Tour;  BEGIN  DBMS\_OUTPUT.ENABLE (buffer\_size => null);  FOR l\_tr IN c\_tr  LOOP  a\_tour\_id := l\_tr.tour\_id;  a\_tour\_name := l\_tr.tour\_name;  a\_tour\_name := l\_tr.explanation;  a\_transport\_id := l\_tr.transport\_id;  a\_period := l\_tr.period;  DBMS\_OUTPUT.PUT\_LINE (a\_tour\_id ||' | '|| a\_tour\_name ||' | '|| a\_tour\_name ||' | '|| a\_transport\_id ||' | '|| a\_period);  END LOOP;  END;  EXCEPTION  WHEN OTHERS  THEN  DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  ----TRANSPORT----  CREATE OR REPLACE PROCEDURE ALL\_TRANSPORT  AS  BEGIN  DECLARE  a\_transport\_id Transport.transport\_id%TYPE;  a\_transport\_name Transport.transport\_name%TYPE;  CURSOR c\_trns IS SELECT \* FROM Transport;  BEGIN  DBMS\_OUTPUT.ENABLE (buffer\_size => null);  FOR l\_trns IN c\_trns  LOOP  a\_transport\_id := l\_trns.transport\_id;  a\_transport\_name := l\_trns.transport\_name;  DBMS\_OUTPUT.PUT\_LINE (a\_transport\_id ||' | '|| a\_transport\_name);  END LOOP;  END;  EXCEPTION  WHEN OTHERS  THEN  DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  ----HOTEL----  CREATE OR REPLACE PROCEDURE ALL\_HOTEL  AS  BEGIN  DECLARE  a\_hotel\_id Hotel.hotel\_id%TYPE;  a\_location\_id Hotel.location\_id%TYPE;  a\_cost\_for\_day Hotel.cost\_for\_day%TYPE;  a\_hotel\_name Hotel.hotel\_name%TYPE;  CURSOR c\_htl IS SELECT \* FROM Hotel;  BEGIN  DBMS\_OUTPUT.ENABLE (buffer\_size => null);  FOR l\_htl IN c\_htl  LOOP  a\_hotel\_id := l\_htl.hotel\_id;  a\_location\_id := l\_htl.location\_id;  a\_cost\_for\_day := l\_htl.cost\_for\_day;  a\_hotel\_name := l\_htl.hotel\_name;  DBMS\_OUTPUT.PUT\_LINE (a\_hotel\_id ||' | '|| a\_location\_id ||' | '|| a\_cost\_for\_day ||' | '|| a\_hotel\_name);  END LOOP;  END;  EXCEPTION  WHEN OTHERS  THEN  DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  ----PLACE----  CREATE OR REPLACE PROCEDURE ALL\_PLACE  AS  BEGIN  DECLARE  a\_location\_id Place.location\_id%TYPE;  a\_country\_name Place.country\_name%TYPE;  a\_city\_name Place.city\_name%TYPE;  CURSOR c\_plc IS SELECT \* FROM Place;  BEGIN  DBMS\_OUTPUT.ENABLE (buffer\_size => null);  FOR l\_plc IN c\_plc  LOOP  a\_location\_id := l\_plc.location\_id;  a\_country\_name := l\_plc.country\_name;  a\_city\_name := l\_plc.city\_name;  DBMS\_OUTPUT.PUT\_LINE (a\_location\_id ||' | '|| a\_country\_name ||' | '|| a\_city\_name);  END LOOP;  END;  EXCEPTION  WHEN OTHERS  THEN  DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  ----MANAGERS----  CREATE OR REPLACE PROCEDURE ALL\_MANAGERS  AS  BEGIN  DECLARE  a\_manager\_id Managers.manager\_id%TYPE;  a\_manager\_firstname Managers.manager\_firstname%TYPE;  a\_manager\_secondname Managers.manager\_secondname%TYPE;  a\_manager\_fathertname Managers.manager\_fathertname%TYPE;  a\_salary Managers.salary%TYPE;  CURSOR c\_mng IS SELECT \* FROM Managers;  BEGIN  DBMS\_OUTPUT.ENABLE (buffer\_size => null);  FOR l\_mng IN c\_mng  LOOP  a\_manager\_id := l\_mng.manager\_id;  a\_manager\_firstname := l\_mng.manager\_firstname;  a\_manager\_secondname := l\_mng.manager\_secondname;  a\_manager\_fathertname := l\_mng.manager\_fathertname;  a\_salary := l\_mng.salary;  DBMS\_OUTPUT.PUT\_LINE (a\_manager\_id ||' | '|| a\_manager\_firstname ||' | '|| a\_manager\_secondname ||' | '|| a\_manager\_fathertname ||' | '|| a\_salary);  END LOOP;  END;  EXCEPTION  WHEN OTHERS  THEN  DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  ----USERS----  CREATE OR REPLACE PROCEDURE ALL\_USERS  AS  BEGIN  DECLARE  a\_user\_id Users.user\_id%TYPE;  a\_user\_firstname Users.user\_firstname%TYPE;  a\_user\_secondname Users.user\_secondname%TYPE;  a\_user\_fathertname Users.user\_fathertnam%TYPE;  a\_user\_age Users.user\_age%TYPE;  CURSOR c\_us IS SELECT \* FROM Users;  BEGIN  DBMS\_OUTPUT.ENABLE (buffer\_size => null);  FOR l\_us IN c\_us  LOOP  a\_user\_id := l\_us.user\_id;  a\_user\_firstname := l\_us.user\_firstname;  a\_user\_secondname := l\_us.user\_secondname;  a\_user\_fathertname := l\_us.user\_fathertname;  a\_user\_age := l\_us.user\_age;  DBMS\_OUTPUT.PUT\_LINE (a\_user\_id ||' | '|| a\_user\_firstname ||' | '|| a\_user\_secondname ||' | '|| a\_user\_fathertname ||' | '|| a\_user\_age);  END LOOP;  END;  EXCEPTION  WHEN OTHERS  THEN  DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  DROP PROCEDURE ALL\_VOUCHER;  DROP PROCEDURE ALL\_TOUR;  DROP PROCEDURE ALL\_TRANSPORT;  DROP PROCEDURE ALL\_HOTEL;  DROP PROCEDURE ALL\_PLACE;  DROP PROCEDURE ALL\_MANAGERS;  DROP PROCEDURE ALL\_USERS;  EXEC ALL\_VOUCHER;  EXEC ALL\_TOUR;  EXEC ALL\_TRANSPORT;  EXEC ALL\_HOTEL;  EXEC ALL\_PLACE;  EXEC ALL\_MANAGERS;  EXEC ALL\_USERS;  -------------------------------DELETE------------------  -------------VOUCHER----------  CREATE OR REPLACE PROCEDURE DELETE\_VOUCHER (  d\_voucher\_id Voucher.voucher\_id%TYPE  )  AS  BEGIN  DELETE FROM Voucher  WHERE Voucher.voucher\_id = d\_voucher\_id;  EXCEPTION  WHEN OTHERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  -------------TOUR----------  CREATE OR REPLACE PROCEDURE DELETE\_TOUR (  d\_tour\_id Tour.tour\_id%TYPE  )  AS  BEGIN  DELETE FROM Tour  WHERE Tour.tour\_id = d\_tour\_id;  EXCEPTION  WHEN OTHERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  -------------TRANSPORT----------  CREATE OR REPLACE PROCEDURE DELETE\_TRANSPORT (  d\_transport\_id Transport.transport\_id%TYPE  )  AS  BEGIN  DELETE FROM Transport  WHERE Transport.transport\_id = d\_transport\_id;  EXCEPTION  WHEN OTHERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  -------------HOTEL----------  CREATE OR REPLACE PROCEDURE DELETE\_HOTEL (  d\_hotel\_id Hotel.hotel\_id%TYPE  )  AS  BEGIN  DELETE FROM Hotel  WHERE Hotel.hotel\_id = d\_hotel\_id;  EXCEPTION  WHEN OTHERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  -------------PLACE----------  CREATE OR REPLACE PROCEDURE DELETE\_PLACE (  d\_location\_id Place.location\_id%TYPE  )  AS  BEGIN  DELETE FROM Place  WHERE Place.location\_id = d\_location\_id;  EXCEPTION  WHEN OTHERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  -------------MANAGERS---------  CREATE OR REPLACE PROCEDURE DELETE\_MANAGERS (  d\_manager\_id Managers.manager\_id%TYPE  )  AS  BEGIN  DELETE FROM Managers  WHERE Managers.manager\_id = d\_manager\_id;  EXCEPTION  WHEN OTHERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  -------------USERS---------  CREATE OR REPLACE PROCEDURE DELETE\_USERS (  d\_user\_id Users.user\_id%TYPE  )  AS  BEGIN  DELETE FROM Users  WHERE Users.user\_id = d\_user\_id;  EXCEPTION  WHEN OTHERS  THEN  SYS.DBMS\_OUTPUT.PUT\_LINE('code error: ' || sqlcode || ' msg oracle: ' || sqlerrm);  END;  DROP PROCEDURE DELETE\_VOUCHER;  DROP PROCEDURE DELETE\_TOUR;  DROP PROCEDURE DELETE\_TRANSPORT;  DROP PROCEDURE DELETE\_HOTEL;  DROP PROCEDURE DELETE\_PLACE;  DROP PROCEDURE DELETE\_MANAGERS;  DROP PROCEDURE DELETE\_USERS;  ----------------------------VIEWS------------------------  CREATE VIEW COST\_HOTEL\_INFO AS  SELECT loc.country\_name, loc.city\_name, hot.hotel\_name, hot.cost\_for\_day  FROM Hotel hot, Place loc WHERE hot.location\_id = loc.location\_id  ORDER BY hot.cost\_for\_day DESC;  CREATE VIEW TOURS\_FOR\_ANY\_WEEKS AS  SELECT \* FROM Tours  WHERE period > 7;  CREATE VIEW VOUCHER\_INFO  AS  SELECT  Voucher.voucher\_id,  Voucher.tour\_date,  Voucher.tour\_id,  Tour.tour\_name,  Transport.transport\_name,  Tour.period,  Tour.explanation,  Voucher.user\_id,  Users.user\_firstname,  Users.user\_secondname,  Users.user\_age,  Managers.manager\_firstname,  Managers.manager\_secondname,  Voucher.hotel\_id,  Hotel.location\_id,  Place.country\_name,  Place.city\_name,  Hotel.cost\_for\_day,  Tour.period \* Hotel.cost\_for\_day AS TO\_PAY  FROM Voucher INNER JOIN Tour ON Voucher.tour\_id = Tour.tour\_id  INNER JOIN Transport ON Tour.transport\_id = Transport.transport\_id  INNER JOIN Users ON Voucher.user\_id = Users.user\_id  INNER JOIN Managers ON Voucher.manager\_id = Managers.manager\_id  INNER JOIN Hotel ON Voucher.hotel\_id = Hotel.hotel\_id  INNER JOIN Place ON Hotel.location\_id = Place.location\_id;  DROP VIEW COST\_HOTEL\_INFO;  DROP VIEW TOURS\_FOR\_ANY\_WEEKS;  DROP VIEW VOUCHER\_INFO;  ---------------------------TRIGGERS----------------------  CREATE OR REPLACE TRIGGER PREVENT\_TABLE\_DELETION  BEFORE DROP ON DATABASE  DECLARE  table\_name VARCHAR2(30);  BEGIN  SELECT object\_name INTO table\_name  FROM user\_objects  WHERE object\_type = 'TABLE' AND object\_name = UPPER(TRIM(SYS\_CONTEXT('USERENV', 'CURRENT\_SCHEMA')));  IF table\_name IS NOT NULL THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Deleting tables is not allowed!');  END IF;  END;  CREATE OR REPLACE TRIGGER CHECK\_USER\_AGE  BEFORE INSERT OR UPDATE ON Users  FOR EACH ROW  BEGIN  IF :NEW.user\_age < 10 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Age cannot be less than 10 years');  END IF;  END;  CREATE OR REPLACE TRIGGER UPDATE\_MANAGER\_SALARY  AFTER INSERT OR UPDATE OF manager\_id ON Voucher  FOR EACH ROW  BEGIN  UPDATE Managers SET salary = salary \* 1.1 WHERE id = :new.manager\_id;  END;  DROP TRIGGER PREVENT\_TABLE\_DELETION;  DROP TRIGGER CHECK\_USER\_AGE;  DROP TRIGGER UPDATE\_MANAGER\_SALARY;  ALTER TRIGGER PREVENT\_TABLE\_DELETION DISABLE;  ALTER TRIGGER PREVENT\_TABLE\_DELETION ENABLE;  ALTER TRIGGER CHECK\_USER\_AGE DISABLE;  ALTER TRIGGER CHECK\_USER\_AGE ENABLE;  ALTER TRIGGER UPDATE\_MANAGER\_SALARY DISABLE;  ALTER TRIGGER UPDATE\_MANAGER\_SALARY ENABLE;  -----------------------------XML----------------  ----DIRECTORY----  CREATE DIRECTORY XML\_DIR AS 'F:\XML';  ----CREATE EXPORT\_TO\_XML----  CREATE OR REPLACE FUNCTION export\_to\_xml(p\_file\_name VARCHAR2) RETURN BOOLEAN  IS  l\_ctx DBMS\_XMLGEN.ctxhandle;  l\_xml CLOB;  l\_file UTL\_FILE.file\_type;  BEGIN  -- Инициализация контекста  l\_ctx := DBMS\_XMLGEN.newcontext('SELECT \* FROM Users');    -- Получение XML-строки  l\_xml := DBMS\_XMLGEN.getxml(l\_ctx);    -- Закрытие контекста  DBMS\_XMLGEN.closecontext(l\_ctx);    -- Создание XML-файла и запись данных в него  l\_file := UTL\_FILE.fopen('XML\_DIR', p\_file\_name, 'w', 32767);  UTL\_FILE.put\_line(l\_file, l\_xml);  UTL\_FILE.fclose(l\_file);    RETURN TRUE;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.put\_line('Error: ' || SQLERRM);  RETURN FALSE;  END;  DECLARE  l\_result BOOLEAN;  BEGIN  l\_result := export\_to\_xml('XMLFILE.xml');  IF l\_result THEN  DBMS\_OUTPUT.put\_line('Export successful');  ELSE  DBMS\_OUTPUT.put\_line('Export failed');  END IF;  END;  ----CREATE  CREATE GLOBAL TEMPORARY TABLE temp\_users  (  user\_id NUMBER,  user\_firstname VARCHAR2(255) not null,  user\_secondname VARCHAR2(255) not null,  user\_fathertname VARCHAR2(255),  user\_age NUMBER  ) ON COMMIT PRESERVE ROWS;  CREATE OR REPLACE PROCEDURE import\_xml\_file  AS  l\_xml\_raw RAW(32767);  l\_xml CLOB;  l\_file UTL\_FILE.file\_type;  BEGIN  -- Открытие XML-файла  l\_file := UTL\_FILE.fopen('XML\_DIR', 'XMLFILE.xml', 'r', 32767);  -- Чтение XML-файла в RAW-переменную  UTL\_FILE.get\_raw(l\_file, l\_xml\_raw, DBMS\_LOB.getlength(l\_xml\_raw));  -- Закрытие файла  UTL\_FILE.fclose(l\_file);  -- Преобразование RAW-данных в CLOB  l\_xml := UTL\_RAW.cast\_to\_varchar2(l\_xml\_raw);  -- Вставка данных во временную таблицу  INSERT INTO temp\_users  SELECT \*  FROM XMLTable('/ROWSET/ROW'  PASSING XMLTYPE(l\_xml)  COLUMNS user\_id NUMBER PATH 'USER\_ID',  user\_firstname VARCHAR2(255) PATH 'USER\_FIRSTNAME',  user\_secondname VARCHAR2(255) PATH 'USER\_SECONDNAME',  user\_fathertname VARCHAR2(255) PATH 'USER\_FATHERTNAME',  user\_age NUMBER PATH 'USER\_AGE');  END;  EXEC import\_xml\_file;  SELECT \* FROM temp\_users |

Листинг 4 – Процедуры и функции