МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 05 01 Информационные системы и технологии

Специализация Базы данных

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«База данных для программного средства «Физкультурно-оздоровительный центр» с применением технологии средств мониторинга состояния СУБД.»

Выполнил студент Кальчевский Даниил Андреевич

(Ф.И.О.)

Руководитель проекта асс. Н.И. Уласевич

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

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

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

Консультант: асс. Н.И. Уласевич.

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

Нормоконтролер: асс. Н.И. Уласевич.

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

Курсовой проект защищен с оценкой

Оглавление

[Введение 6](#_Toc135009580)

[1. Постановка задачи 7](#_Toc135009581)

[1.1. Техническое задание 7](#_Toc135009582)

[1.2. Описание используемых технологий 7](#_Toc135009583)

[2. Проектирование базы данных 9](#_Toc135009584)

[2.1. Этапы проектирования базы данных 9](#_Toc135009585)

[2.2. Реализация проектирования базы данных 11](#_Toc135009586)

[2.3. Вывод 17](#_Toc135009587)

[3. Разработка объектов базы данных 19](#_Toc135009588)

[3.1. Хранимые процедуры 19](#_Toc135009589)

[3.2. Функции 20](#_Toc135009590)

[3.3. Представления и материализованные представления 20](#_Toc135009591)

[3.4. Индексы 21](#_Toc135009592)

[3.5. Триггеры 22](#_Toc135009593)

[3.6. Синонимы 23](#_Toc135009594)

[3.7. Вывод 23](#_Toc135009595)

[4. Описание процедур импорта и экспорта 24](#_Toc135009596)

[4.1. Экспорт данных 24](#_Toc135009597)

[4.2. Импорт данных 26](#_Toc135009598)

[4.3. Вывод 28](#_Toc135009599)

[5. Тестирование производительности 29](#_Toc135009600)

[5.3. Вывод 32](#_Toc135009601)

[6. Описание технологии и ее применения в базе данных 33](#_Toc135009602)

[6.1. Описание используемой технологии 33](#_Toc135009603)

[6.3. Вывод 34](#_Toc135009604)

[Заключение 35](#_Toc135009605)

[ПРИЛОЖЕНИЕ А Листинг скриптов создания таблиц 37](#_Toc135009606)

[ПРИЛОЖЕНИЕ Б Листинг Процедуры добавления 40](#_Toc135009607)

[ПРИЛОЖЕНИЕ В Листинг Export всех таблиц 46](#_Toc135009608)

[ПРИЛОЖЕНИЕ Г Листинг Import всех таблиц 49](#_Toc135009609)

[ПРИЛОЖЕНИЕ Д Листинг Синонимы всех таблиц 56](#_Toc135009610)

# Введение

В современном мире многие люди постоянно имеют желание быть здоровым, поддерживать хорошую форму. Для обеспечения получения желаемого продукта, клиенту необходимо провести некоторые процедуры по внесению и получению данных. Эффективным и безопасным типом хранения данных в наше время является хранение информации в цифровом виде. Для более эффективного использования и манипулирования данными используют базы данных. К примеру, для хранения информации о комплексах, продажах и клиентах, необходима база данных.

Целью данной учебно-проектной работы является разработка реляционной базы данных физкультурно-оздоровительного центра. Реализация базы данных должна осуществляться в рамках системы управления базами данных Oracle 12c, с помощью применения технологии мониторинга состояния СУБД.

Для подтверждения актуальности и соответствии поставленным задачам, база данных должна отвечать следующим требованиям:

* инфраструктура базы данных должна быть приведена к третьей нормальной форме 3INF;
* информация базы данных должна быть корректна и осмыслена;
* возможность получения различных данных, соответствующих каждому уровню доступа;
* осуществление поиска данных по определённым критериям, сортировка и упорядочивание.

В пояснительной записке вы сможете найти краткую информацию о похожих продуктах, архитектуре, реализации проекта, руководстве пользователя.

# 1. Постановка задачи

**1.1. Техническое задание**

Целью данного курсового проекта является проектирование и разработка базы данных для программного средства физкультурно-оздоровительного центра с применением технологии средств мониторинга СУБД. В качестве модели данных будет использована реляционная модель. В качестве среды разработки выбрана система управления базами данных Oracle 12c. Проектирование данных будет производиться в соответствии с общими требованиями.

Функционал разрабатываемой базы данных должен позволять:

* добавление и изменение данных о клиентах;
* добавление и изменение данных о абонементов;
* добавление и изменение данных о продажи абонементов;
* добавление и изменение данных о информации о комплексах;
* добавление и изменение данных о расписании тренировок;
* добавление и изменение данных о списках услуг;
* добавление и изменение данных о типах тренировках;
* добавление и изменение данных о сотрудниках;

Для реализации данного функционала будут разработаны объекты базы данных, такие как:

* роли пользователей;
* таблицы;
* процедуры;
* профили безопасности;
* табличные пространства;
* пользователи;
* представления;
* индексы.

**1.2. Описание используемых технологий**

На данный момент существует много различных систем управления базами данных, такие как: Oracle Database, MS SQL Server, NoSQL, MySql и т.д. Для разработки данной учебно-проектной работы была выбрана такая СУБД, как «Oracle 12c», так как посредством использования её инструментов, возможно выполнить все поставленные требования в рамках курсового проекта.

Касательно используемой технологии базы данных, была выбрана технология мониторинга состояния СУБД.

Технология мониторинга СУБД (системы управления базами данных) предназначена для наблюдения за работой баз данных и выявления возможных проблем. Она позволяет администраторам баз данных получать информацию о производительности, нагрузке, использовании ресурсов и других параметрах, которые могут влиять на работу баз данных.

Основными инструментами мониторинга СУБД являются системы управления мониторингом, которые могут быть интегрированы в СУБД или работать отдельно. Они могут собирать данные о производительности, нагрузке, использовании ресурсов и других параметрах, а затем анализировать их для выявления проблем.

Системы мониторинга СУБД могут также предоставлять администраторам баз данных информацию о состоянии баз данных, включая ошибки, блокировки и другие проблемы. Это позволяет администраторам быстро реагировать на проблемы и предотвращать их возникновение.

В целом, технология мониторинга СУБД является важным инструментом для обеспечения надежной и эффективной работы баз данных. Она позволяет администраторам баз данных быстро выявлять и решать проблемы, что способствует повышению производительности и улучшению качества обслуживания пользователей.

* 1. **Вывод**

В данном разделе было произведено анализирование и описание аналогичных решений в сфере разработки учебно-проектной работы.

Была сформирована и описана поставленная задача для разработки учебно-проектной работы. Были определены технические инструменты для выполнения данной работы, а также выбрана технология, применяемая к разрабатываемой базе данных.

**2. Проектирование базы данных**

**2.1. Этапы проектирования базы данных**

Процесс проектирования базы данных состоит из следующих этапов:

* сбор информации;
* определение сущностей;
* определение атрибутов для каждой сущности;
* определение связей между сущностями;
* нормализация.

На этапе сбора информации вам необходимо точно определить, как будет использоваться база данных, и какая информация будет в ней храниться.

Далее следует этап определения сущностей и на нем определяются сущности, из которых будет состоять база данных.

Сущность — это объект в базе данных, в котором хранятся данные. Сущность может представлять собой нечто вещественное (дом, человек, предмет, место) или абстрактное (банковская операция, отдел компании, маршрут автобуса). В физической модели сущность называется таблицей. Сущности состоят из атрибутов (столбцов таблицы) и записей (строк в таблице).

Обычно базы данных состоят из нескольких основных сущностей, связанных с большим количеством подчиненных сущностей. Основные сущности называются независимыми: они не зависят ни от какой-либо другой сущности. Подчиненные сущности называются зависимыми: для того, чтобы существовала одна из них, должна существовать связанная с ней основная таблица.

Следующий этап — это определение атрибутов созданных сущностей. Атрибут представляет свойство, описывающее сущность. Атрибуты часто бывают числом, датой или текстом. Все данные, хранящиеся в атрибуте, должны иметь одинаковый тип и обладать одинаковыми свойствами.

В физической модели атрибуты называют колонками. После определения сущностей необходимо определить все атрибуты этих сущностей. На диаграммах атрибуты обычно перечисляются внутри прямоугольника сущности.

На этапе определения связей работа происходит с ключами сущностей. Ключом называется набор атрибутов, однозначно определяющий запись. Ключи делятся на два класса: простые и составные.

Простой ключ состоит только из одного атрибута. Например, в базе «Паспорта граждан страны» номер паспорта будет простым ключом: ведь не бывает двух паспортов с одинаковым номером.

Составной ключ состоит из нескольких атрибутов. В той же базе «Паспорта граждан страны» может быть составной ключ со следующими атрибутами:  
фамилия, имя, отчество, дата рождения. Это — как пример, т. к. этот составной ключ, теоретически, не обеспечивает гарантированной уникальности записи.

Первичным ключом называется совокупность атрибутов, однозначно идентифицирующих запись в таблице (сущности). Один из возможных ключей становится первичным ключом. На диаграммах первичные ключи часто изображаются выше основного списка атрибутов или выделяются специальными символами.

Любой возможный ключ, не являющийся первичным, называется альтернативным ключом. Сущность может иметь несколько альтернативных ключей. Внешним ключом называется совокупность атрибутов, ссылающихся на первичный или альтернативный ключ другой сущности. Если внешний ключ не связан с первичной сущностью, то он может содержать только неопределенные значения. Если при этом ключ является составным, то все атрибуты внешнего ключа должны быть неопределенными. На диаграммах атрибуты, объединяемые во внешние ключи, обозначаются специальными символами.

Реляционные базы данных позволяют объединять информацию, принадлежащую разным сущностям. Отношение — это ситуация, при которой одна сущность ссылается на первичный ключ второй сущности. Они определяются в процессе проектирования базы. Для этого следует проанализировать сущности и выявить логические связи, существующие между ними.

Тип отношения определяет количество записей сущности, связанных с записью другой сущности. Отношения делятся на три основных типа:

* Один-к-одному. Каждой записи первой сущности соответствует только одна запись из второй сущности. А каждой записи второй сущности соответствует только одна запись из первой сущности. Например, есть две сущности: Люди и Свидетельства о рождении. И у одного человека может быть только одно свидетельство о рождении.
* Один-ко-многим. Каждой записи первой сущности могут соответствовать несколько записей из второй сущности. Однако каждой записи второй сущности соответствует только одна запись из первой сущности. Например, есть две сущности: Заказ и Позиция заказа. И в одном заказе может быть много товаров.
* Многие-ко-многим. Каждой записи первой сущности могут соответствовать несколько записей из второй сущности. Однако и каждой записи второй сущности может соответствовать несколько записей из первой сущности. Например, есть две сущности: Автор и Книга. Один автор может написать много книг. Но у книги может быть несколько авторов.

По критерию обязательности отношения делятся на обязательные и необязательные.

* Обязательное отношение означает, что для каждой записи из первой сущности непременно должны присутствовать связанные записи во второй сущности.
* Необязательное отношение означает, что для записи из первой сущности может и не существовать записи во второй сущности.

Далее следует этап нормализации. Нормализацией называется процесс удаления избыточных данных из базы данных. Каждый элемент данных должен храниться в базе в одном и только в одном экземпляре. Существует пять распространенных форм нормализации. Как правило, база данных приводится к третьей нормальной форме.

В процессе нормализации выполняются определенные действия по удалению избыточных данных. Нормализация повышает быстродействие, ускоряет сортировку и построение индекса, уменьшает количество индексов на сущность, ускоряет операции вставки и обновления.

Нормализованная база данных обычно отличается большей гибкостью. При модификации запросов или сохраняемых данных в нормализованную базу обычно приходится вносить меньше изменений, а внесение изменений имеет меньше последствий.

Чтобы преобразовать сущность в первую нормальную форму, следует исключить повторяющиеся группы значений и добиться того, чтобы каждый атрибут содержал только одно значение, списки значений не допускаются.  
Другими словами, каждый атрибут, в сущности, должен храниться только в одном экземпляре.

Для соответствия второй нормальной форме сущности должны быть в первой нормальной форме. Таблица во второй нормальной форме содержит только те данные, которые к ней относятся. Значения не ключевых атрибутов сущности зависят от первичного ключа. Если более точно, то атрибуты зависят от первичного ключа, от всего первичного ключа и только от первичного ключа.

В третьей нормальной форме исключаются атрибуты, не зависящие от всего ключа. Любая сущность, находящаяся в третьей нормальной форме, находится также и во второй. Это самая распространенная форма базы данных.  
В третьей нормальной форме каждый атрибут зависит от ключа, от всего ключа и ни от чего, кроме ключа.

Ограничения — это правила, за соблюдением которых следит система управления базы данных. Ограничения определяют множество значений, которые можно вводить в столбец или столбцы.

Организовав данные в таблицы и определив связи между ними, можно считать, что была создана модель, правильным образом отражающая бизнес-среду. Теперь нужно обеспечить, чтобы данные, вводимые в базу, давали правильное представление о состоянии дела. Иными словами, нужно обеспечить выполнение деловых правил и поддержку целостности базы данных.

Например, ваша компания занимается доставкой книг. Вы вряд ли примете заказ от неизвестного клиента, ведь тогда вы даже не сможете доставить заказ. Отсюда бизнес-правило: заказы принимаются только от клиентов, информация о которых есть в базе данных.

Корректность данных в реляционных базах обеспечивается набором правил. Правила целостности данных делятся на четыре категории.

**2.2. Реализация проектирования базы данных**

Для реализации необходимого функционала была создана база данных, таблицы которой можно разделить на 3 логические группы: таблицы для хранения информации о продуктах, таблицы для хранения информации о заказах пользователей и таблицы для хранения пользовательских данных.

Рассмотрим таблицы базы данных на примере диаграммы, представленной на рисунке 2.1.

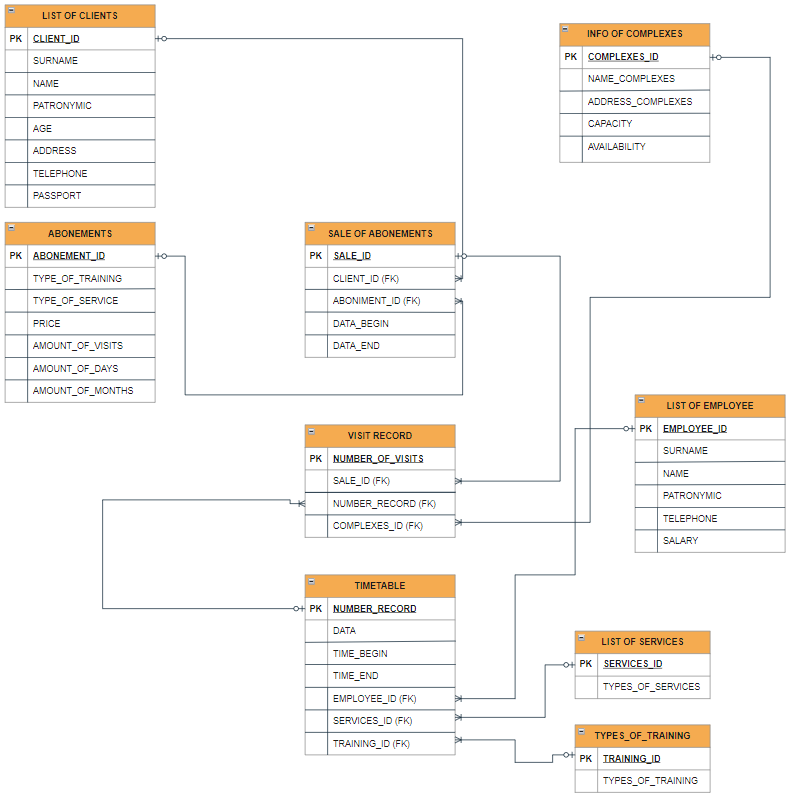


Рисунок 2.1 – Диаграмма таблиц базы данных

База данных состоит из 9 таблиц, связанных между собой внешними ключами.

Таблица “LIST\_OF\_CLIENT” хранит информацию о клиентах, включая их имя, возраст, адрес и контактную форму.

ID\_CLIENT: числовой столбец, который генерируется по умолчанию как идентификатор, начиная с 1 и увеличиваясь на 1 для каждой новой строки. Этот столбец не может содержать пустые значения.

SURNAME\_CLIENT: строковый столбец, который хранит фамилию клиента. Этот столбец не может содержать пустые значения и имеет максимальную длину 50 символов.

NAME\_CLIENT: строковый столбец, который хранит имя клиента. Этот столбец не может содержать пустые значения и имеет максимальную длину 50 символов.

PATRONYMIC\_CLIENT: строковый столбец, который хранит отчество клиента. Этот столбец не может содержать пустые значения и имеет максимальную длину 50 символов.

AGE\_CLIENT: числовой столбец, который хранит возраст клиента. Этот столбец не может содержать пустые значения.

ADDRESS\_CLIENT: строковый столбец, который хранит адрес клиента. Этот столбец не может содержать пустые значения и имеет максимальную длину 200 символов.

TELEPHONE\_CLIENT: строковый столбец, который хранит номер телефона клиента. Этот столбец не может содержать пустые значения и имеет максимальную длину 13 символов.

PASSPORT\_CLIENT: строковый столбец, который хранит номер паспорта клиента. Этот столбец не может содержать пустые значения и имеет максимальную длину 20 символов.

Таблица также имеет ограничение первичного ключа с именем "PK\_LIST\_OF\_CLIENTS", которое основано на столбце ID\_CLIENT. Это гарантирует, что каждая строка в таблице имеет уникальный идентификатор.

Таблица “ABONEMENTS” хранит информацию о различных видах тренировочных услуг, предлагаемых комплексом, включая цену, количество посещений, количество дней и месяцев.

"ID\_ABONEMENT" - уникальный идентификатор для каждого абонемента, генерируемый автоматически с помощью ключевого слова "IDENTITY". Столбец имеет тип "NUMBER" и не может быть пустым.

"TYPE\_OF\_TRAINING" - строковый столбец, который указывает тип тренировки, предлагаемой абонементом. Столбец имеет тип "NVARCHAR2" и не может быть пустым.

"TYPE\_OF\_SERVICE" - строковый столбец, который указывает тип услуги, предлагаемой абонементом. Столбец имеет тип "NVARCHAR2" и не может быть пустым.

"PRICE" - числовой столбец, который указывает цену абонемента. Столбец имеет тип "NUMBER" и не может быть пустым.

"AMOUNT\_OF\_VISITS" - числовой столбец, который указывает количество посещений, включенных в абонемент. Столбец имеет тип "NUMBER" и не может быть пустым

"AMOUNT\_OF\_DAYS" - числовой столбец, который указывает количество дней, на которые действует абонемент. Столбец имеет тип "NUMBER" и не может быть пустым.

"AMOUNT\_OF\_MONTHS" - числовой столбец, который указывает количество месяцев, на которые действует абонемент. Столбец имеет тип "NUMBER" и не может быть пустым.

Таблица также имеет ограничение первичного ключа на столбец "ID\_ABONEMENT", которое гарантирует, что каждая строка в таблице имеет уникальный идентификатор.

Таблица “SALE\_OF\_ABONEMENTS” хранит информацию о продаже абонементов клиентам, включая даты начала и окончания абонементов.

ID\_SALE: уникальный идентификатор продажи, который генерируется автоматически при добавлении новой записи в таблицу.

ID\_CLIENT: идентификатор клиента, который приобрел абонемент. Это обязательное поле и ссылается на таблицу LIST\_OF\_CLIENTS по внешнему ключу FK\_SALE\_OF\_ABONEMENTS.

ID\_ABONEMENT: идентификатор абонемента, который был продан. Это обязательное поле и ссылается на таблицу ABONEMENTS по внешнему ключу FK\_SALE\_OF\_ABONEMENTS2.

DATA\_BEGIN: дата начала действия абонемента. Это обязательное поле.

DATA\_END: дата окончания действия абонемента. Это обязательное поле.

Также в таблице есть два ограничения:

PK\_SALE\_OF\_ABONEMENTS: это первичный ключ таблицы, который гарантирует уникальность каждой записи в таблице.

FK\_SALE\_OF\_ABONEMENTS: это внешний ключ, который связывает столбец ID\_CLIENT с таблицей LIST\_OF\_CLIENTS и гарантирует, что каждый клиент, который приобрел абонемент, существует в таблице LIST\_OF\_CLIENTS.

FK\_SALE\_OF\_ABONEMENTS2: это внешний ключ, который связывает столбец ID\_ABONEMENT с таблицей ABONEMENTS и гарантирует, что каждый проданный абонемент существует в таблице ABONEMENTS.

Таблица “INFO\_COMPLEXES” хранит информацию о спортивных комплексах, включая их название, адрес и вместимость.

ID\_COMPLEXES: числовой столбец, который генерируется автоматически при вставке новой записи в таблицу. Он уникален для каждой записи и не может быть NULL.

NAME\_COMPLEXES: строковый столбец, который содержит название комплекса. Максимальная длина этого столбца составляет 100 символов.

ADDRESS\_COMPLEXES: строковый столбец, который содержит адрес комплекса. Максимальная длина этого столбца составляет 100 символов.

CAPACITY\_COMPLEXES: числовой столбец, который содержит вместимость комплекса. Он может быть NULL.

PK\_INFO\_COMPLEXES: это ограничение первичного ключа, которое гарантирует уникальность значений в столбце ID\_COMPLEXES.

Таблица “VISIT\_RECORD” хранит информацию о посещении клиентом оздоровительного центра(комплекса), включая дату и время посещения, номер записи, номер комплекса.

NUMBER\_OF\_VISITS - столбец типа NUMBER, который генерируется автоматически при вставке новой записи в таблицу. Значения этого столбца начинаются с 1 и увеличиваются на 1 при каждой новой записи. Данный столбец не может быть пустым и является первичным ключом таблицы.

ID\_SALE - столбец типа NUMBER, который содержит идентификатор продажи абонементов. Данный столбец не может быть пустым и связан с таблицей SALE\_OF\_ABONEMENTS по внешнему ключу.

NUMBER\_RECORD - столбец типа NUMBER, который содержит номер записи в расписании занятий. Данный столбец не может быть пустым и связан с таблицей TIME\_TABLE по внешнему ключу.

ID\_COMPLEXES - столбец типа NUMBER, который содержит идентификатор комплекса, в котором проходит занятие. Данный столбец не может быть пустым и связан с таблицей INFO\_COMPLEXES по внешнему ключу.

Также в таблице присутствуют следующие ограничения:

PK\_VISIT\_RECORD - первичный ключ таблицы, который состоит из столбца NUMBER\_OF\_VISITS.

FK\_VISIT\_RECORD - внешний ключ, который связывает столбец ID\_SALE с таблицей SALE\_OF\_ABONEMENTS.

FK\_VISIT\_RECORD2 - внешний ключ, который связывает столбец NUMBER\_RECORD с таблицей TIME\_TABLE.

FK\_VISIT\_RECORD3 - внешний ключ, который связывает столбец ID\_COMPLEXES с таблицей INFO\_COMPLEXES.

Таблица “TIME\_TABLE” хранит информацию о расписании оздоровительного центра, включая дату и время каждой тренировки, сотрудника, предоставляющего услугу, и тип предлагаемой тренировки.

NUMBER\_RECORD - столбец типа NUMBER, который генерируется автоматически при добавлении новой записи в таблицу. Значения этого столбца начинаются с 1 и увеличиваются на 1 при каждой новой записи. Этот столбец не может быть пустым.

DATA\_TIME\_TABLE - столбец типа DATE, который содержит дату записи в таблицу. Этот столбец не может быть пустым.

TIME\_BEGIN - столбец типа NVARCHAR2(20), который содержит время начала работы. Этот столбец не может быть пустым.

TIME\_END - столбец типа NVARCHAR2(20), который содержит время окончания работы. Этот столбец не может быть пустым.

ID\_EMPLOYEE - столбец типа NUMBER, который содержит идентификатор сотрудника. Этот столбец не может быть пустым и связан с таблицей LIST\_OF\_EMPLOYEE по внешнему ключу FK\_TIME\_TABLE.

ID\_SERVICES - столбец типа NUMBER, который содержит идентификатор услуги. Этот столбец не может быть пустым и связан с таблицей LIST\_OF\_SERVICES по внешнему ключу FK\_TIME\_TABLE2.

ID\_TRAINING - столбец типа NUMBER, который содержит идентификатор типа тренировки. Этот столбец не может быть пустым и связан с таблицей TYPES\_OF\_TRAINING по внешнему ключу FK\_TIME\_TABLE3.

Также в таблице присутствуют следующие ограничения:

PK\_TIME\_TABLE - первичный ключ таблицы, который состоит из столбца NUMBER\_RECORD.

FK\_TIME\_TABLE - внешний ключ, который связывает столбец ID\_EMPLOYEE с таблицей LIST\_OF\_EMPLOYEE.

FK\_TIME\_TABLE2 - внешний ключ, который связывает столбец ID\_SERVICES с таблицей LIST\_OF\_SERVICES.

FK\_TIME\_TABLE3 - внешний ключ, который связывает столбец ID\_TRAINING с таблицей TYPES\_OF\_TRAINING.

Таблица “LIST\_OF\_SERVICES” хранит информацию о различных типах тренировок, предлагаемых оздоровительным центром.

"ID\_SERVICES" — это уникальный идентификатор услуги, который генерируется автоматически при добавлении новой записи в таблицу. Этот столбец имеет тип данных "NUMBER" и не может быть пустым.

"TYPES\_OF\_SERVICES" — это название типа услуги. Этот столбец имеет тип данных "NVARCHAR2" и может содержать до 50 символов. Он также не может быть пустым.

"PK\_LIST\_OF\_SERVICES" — это ограничение первичного ключа, которое гарантирует уникальность каждой записи в таблице. Оно связано с столбцом "ID\_SERVICES".

Таблица “TYPES\_OF\_TRAINING” хранит информацию о различных типах тренировок, предлагаемых оздоровительным центром.

Столбец ID\_TRAINING имеет тип данных NUMBER и генерируется автоматически при вставке новой записи в таблицу. Значения этого столбца начинаются с 1 и увеличиваются на 1 при каждой последующей вставке. Этот столбец не может содержать NULL-значения и является первичным ключом таблицы.

Столбец TYPES\_OF\_TRAINING имеет тип данных NVARCHAR2 и может содержать до 50 символов. Этот столбец также не может содержать NULL-значения и содержит информацию о типах тренировок.

Таблица “LIST\_OF\_EMPLOYEE” хранит информацию о сотрудниках оздоровительного центра, включая их имена, контактную информацию и зарплату.

"ID\_EMPLOYEE" - уникальный идентификатор сотрудника, который генерируется автоматически при добавлении новой записи в таблицу. Этот столбец имеет тип данных "NUMBER".

"SURNAME\_EMPLOYEE" - фамилия сотрудника. Этот столбец имеет тип данных "NVARCHAR2" и длину 50 символов.

"NAME\_EMPLOYEE" - имя сотрудника. Этот столбец имеет тип данных "NVARCHAR2" и длину 50 символов.

"PATRONYMIC\_EMPLOYEE" - отчество сотрудника. Этот столбец имеет тип данных "NVARCHAR2" и длину 50 символов.

"TELEPHONE\_EMPLOYEE" - номер телефона сотрудника. Этот столбец имеет тип данных "NVARCHAR2" и длину 13 символов.

"ADDRESS\_EMPLOYEE" - адрес сотрудника. Этот столбец имеет тип данных "NVARCHAR2" и длину 50 символов.

"SALARY\_EMPLOYEE" - зарплата сотрудника. Этот столбец имеет тип данных "NUMBER".

Также в таблице есть ограничение "PK\_LIST\_OF\_EMPLOYEE", которое задает первичный ключ для таблицы и гарантирует уникальность значений в столбце "ID\_EMPLOYEE".

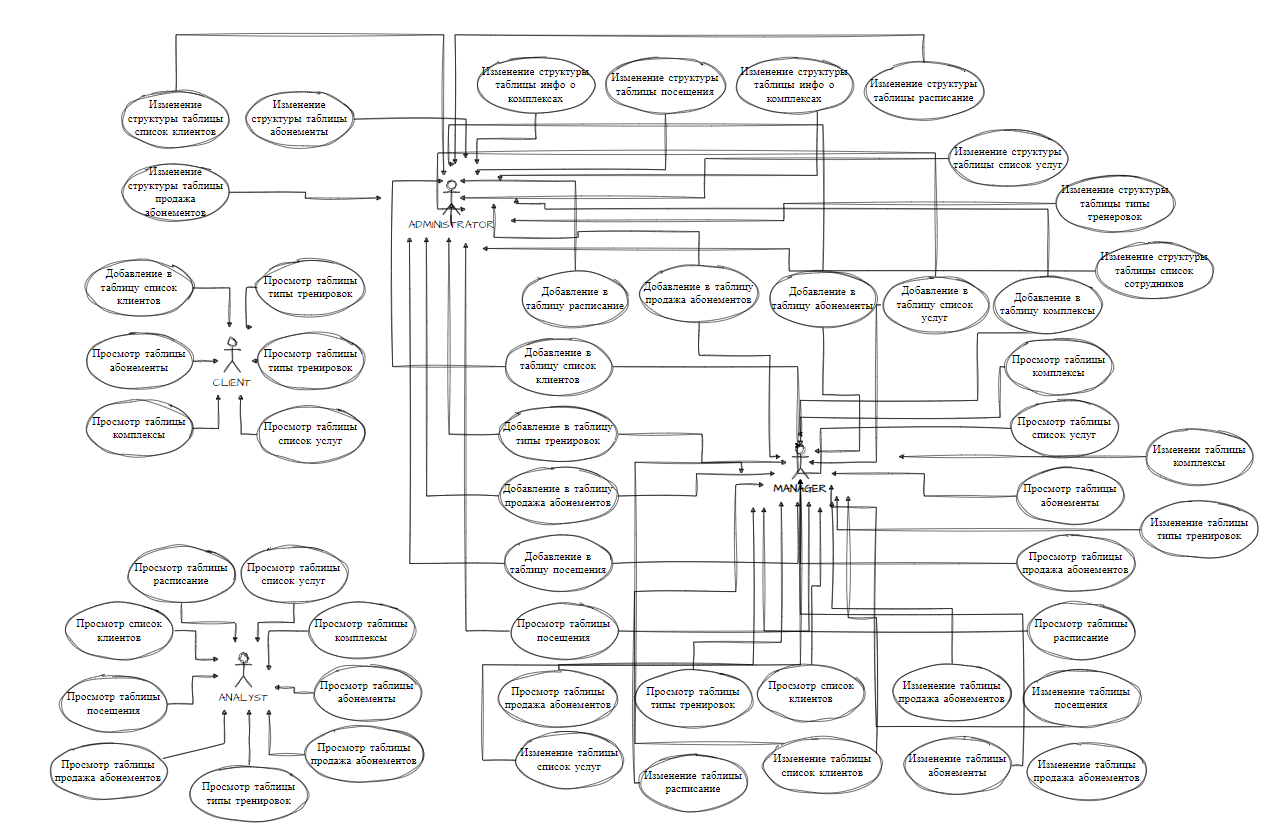


Рисунок 2.2 – Диаграмма вариантов использования

Диаграмма вариантов использования (Use Case Diagram) — это графическое представление того, как различные пользователи (актеры) будут использовать систему для достижения своих целей, показывая функциональные требования к системе и ее взаимодействие с внешними пользователями и другими системами.

Основное назначение диаграммы вариантов использования заключается в том, чтобы помочь разработчикам и заказчикам понять, как система будет использоваться в реальном мире. Она позволяет определить функциональные требования к системе и убедиться, что все необходимые функции будут реализованы. Кроме того, диаграмма вариантов использования может использоваться для создания тестовых сценариев и обучения пользователей, как использовать систему.

**2.3. Вывод**

Проектирование базы данных — это процесс создания структуры базы данных, которая будет использоваться для хранения, управления и обработки данных. Цель проектирования базы данных - создать эффективную и надежную структуру данных, которая будет соответствовать требованиям бизнеса и обеспечивать быстрый доступ к данным.

При проектировании базы данных необходимо учитывать следующие факторы:

* требования бизнеса: необходимо понимать, какие данные будут храниться в базе данных, как они будут использоваться и какие операции будут выполняться с этими данными;
* нормализация: процесс разделения таблиц на более мелкие и связанные таблицы, чтобы уменьшить дублирование данных и обеспечить целостность данных;
* индексы: индексы используются для ускорения поиска данных в таблицах. Необходимо определить, какие поля будут индексироваться и какие типы индексов будут использоваться;
* ограничения: ограничения используются для обеспечения целостности данных и защиты от ошибок. Необходимо определить, какие ограничения будут использоваться для каждой таблицы;
* схема базы данных: необходимо определить, какие таблицы будут созданы, как они будут связаны между собой и какие поля будут использоваться для связи;
* безопасность: необходимо определить, какие пользователи будут иметь доступ к базе данных и какие права доступа будут у них;
* производительность: необходимо учитывать производительность базы данных при проектировании, чтобы обеспечить быстрый доступ к данным и минимизировать время выполнения запросов.

В целом, проектирование базы данных – это сложный процесс, который требует внимательного анализа требований бизнеса и определения наилучших практик для создания эффективной и надежной структуры данных.

**3. Разработка объектов базы данных**

**3.1. Хранимые процедуры**

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

Для обеспечения взаимодействия с базой данных были разработаны хранимые процедуры вставки, изменения и получения данных.

CREATE OR REPLACE PROCEDURE ADD\_ROWS\_TO\_LIST\_OF\_CLIENTS(pSURNAME\_CLIENT NVARCHAR2, pNAME\_CLIENT NVARCHAR2, pPATRONYMIC\_CLIENT NVARCHAR2, pAGE\_CLIENT NUMBER, pADDRESS\_CLIENT NVARCHAR2, pTELEPHONE\_CLIENT NVARCHAR2, pPASSPORT\_CLIENT NVARCHAR2)

IS

BEGIN

        INSERT INTO LISOFCLI(SURNAME\_CLIENT, NAME\_CLIENT, PATRONYMIC\_CLIENT, AGE\_CLIENT, ADDRESS\_CLIENT, TELEPHONE\_CLIENT, PASSPORT\_CLIENT)

        VALUES(pSURNAME\_CLIENT, pNAME\_CLIENT, pPATRONYMIC\_CLIENT, pAGE\_CLIENT, pADDRESS\_CLIENT, pTELEPHONE\_CLIENT, pPASSPORT\_CLIENT);

        EXCEPTION

            WHEN OTHERS

                THEN DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);

END;

Листинг 3.1 – Пример хранимой процедуры

При разработке курсового проекта было создано большое количество процедур для следующих целей:

Для пользователя:

* добавление личных данных;
* получение списка абонементов;
* получение списка комплексов;
* получение списка услуг;
* получение списка типов тренировок;

Для сотрудника:

* добавление личных данных;
* просмотр данных о услугах;
* просмотр данных о типах тренировок;
* просмотр данных о продажах абонементов;
* просмотр данных о клиентах;
* просмотр данных о сотрудниках;
* изменение данных о клиентах;
* добавление данных о комплексах;
* добавление данных о продажах абонементов;
* добавление данных о клиентах;
* добавление данных о сотрудниках.

**3.2. Функции**

Функция SQL — это набор операторов SQL, которые принимают входные данные и выполняют с ними действия SQL, а затем возвращают результаты в виде выходных данных.

CREATE OR REPLACE FUNCTION COUNT\_LIST\_OF\_CLIENTS

RETURN NUMBER

IS

    count\_clients NUMBER;

BEGIN

    SELECT COUNT(\*) INTO count\_clients FROM LIST\_OF\_CLIENTS;

    RETURN count\_clients;

END;

DECLARE

     count\_clients NUMBER;

BEGIN

    count\_clients := COUNT\_LIST\_OF\_CLIENTS;

    DBMS\_OUTPUT.PUT\_LINE('Number of Clients: ' || ' ' || count\_clients);

END;

Листинг 3.2 – Пример функции

Функции выполняют утилитарные дополнительные функции и служат для упрощенной выборки некоторых часто получаемых данных, так как они позволяют возможность удобно получить некоторые данные:

* Возвращает количество записей в таблице LIST\_OF\_CLIENTS.
* Возвращает количество доступных посещений, которые остались на абонементе из таблицы ABONEMENTS.
* Возвращает список типов тренировок из таблицы TYPES\_OF\_TRAINING.
* Возвращает количество клиентов из таблицы LIST\_OF\_CLIENTS, чей возраст превышает заданный пороговый возраст.

**3.3. Представления и материализованные представления**

Представление – это поименованный select-запрос из таблиц, основанный на результирующем наборе инструкции SQL.

Оно содержит строки и столбцы, как и настоящая таблица. Поля в представлении – это поля из одной или нескольких реальных таблиц в базе данных.

Представления используются для абстракции от соединения таблиц и сокрытия их реализации внутри доступных пользователям процедур.

CREATE VIEW LISOFCLI\_VIEW AS SELECT \* FROM LIST\_OF\_CLIENTS;

CREATE MATERIALIZED VIEW MATER\_VIEW\_LIST\_OF\_SERVICES

BUILD IMMEDIATE

REFRESH ON DEMAND

ENABLE QUERY REWRITE

AS SELECT \* FROM LIST\_OF\_SERVICES;

Листинг 3.3 – Пример материализованного и стандартного представлений

В данном курсовом проекте были созданы представления для каждой из таблиц, которые понадобятся пользователям с ролью “аналитик”, для статистического анализа данных, без возможности их повреждения.

* представление таблицы список клиентов;
* представление таблицы абонементы;
* представление таблицы продажа абонементов;
* представление таблицы информация о комплексах;
* представление таблицы информация о записях посещения;
* представление таблицы расписание;
* представление таблицы список услуг;
* представление таблицы типы тренировок;
* представление таблицы список сотрудников;

Материализованное представление – физический объект базы данных, содержащий результат выполнения запроса. Материализованные представления позволяют многократно ускорить выполнение запросов, обращающихся к большому количеству записей, позволяя за секунды выполнять запросы к терабайтам данных.

В данном курсовом проекте были реализованы 2 материализованных представления для списка услуг и для типов тренировок, так как данные таблицы редко обновляются и удобно хранить подгруженные данные в материализованных представлениях.

* материализованное представление таблицы список услуг;
* материализованное представление таблицы типы тренировок.

**3.4. Индексы**

Индекс – объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска.

CREATE INDEX INDEX\_ID\_CLIENT ON SALE\_OF\_ABONEMENTS(ID\_CLIENT);

Листинг 3.4 – Пример индекса

В данном курсовом проекте были реализованы 7 индексов по таким таблицам, как: продажи абонементов, расписание, список клиентов, посещенные записи:

* индекс в таблице продажа абонементов по столбцу ID\_CLIENT;
* индекс в таблице продажа абонементов по столбцу ID\_ABONEMENT;
* индекс в таблице расписание по столбцу ID\_EMPLOYEE;
* индекс в таблице расписание по столбцу ID\_SERVICES;
* индекс в таблице расписание по столбцу ID\_TRAINING;
* индекс в таблице список клиентов по столбцу SURNAME\_CLIENT;
* индекс в таблице посещенные записи по столбцу ID\_COMPLEXES.

**3.5. Триггеры**

Триггер – хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определённом столбце заданной таблицы реляционной базы данных.

CREATE OR REPLACE TRIGGER PREVENT\_CLIENT\_DELETION

BEFORE DELETE ON LIST\_OF\_CLIENTS

FOR EACH ROW

DECLARE

    active\_abonnements NUMBER;

BEGIN

    SELECT COUNT(\*)

    INTO active\_abonnements

    FROM SALE\_OF\_ABONEMENTS

    WHERE ID\_CLIENT = :OLD.ID\_CLIENT

    AND DATA\_END > SYSDATE;

    IF active\_abonnements > 0 THEN

        RAISE\_APPLICATION\_ERROR(-20001, 'Cannot delete client with active abonnement');

    END IF;

END;

Листинг 3.5 – Пример триггера

В данном курсовом проекте были реализован DDL-триггер на запрет удаления клиента у которого имеется активный абонемент:

* триггер на запрет удаления клиента у которого имеется активный абонемент;

**3.6. Синонимы**

Синонимы – это псевдонимы объектов базы данных, которые служат в основном для облегчения пользователям доступа к объектам, принадлежащим другим пользователям, а также в целях безопасности.

CREATE PUBLIC SYNONYM LISOFCLI FOR KP\_OWNER.LIST\_OF\_CLIENTS;

Листинг 3.6 – Пример синонима

В данном курсовом проекте были реализованы публичные синонимы для всех созданных таблиц базы данных:

* синоним таблицы список клиентов;
* синоним таблицы абонементы;
* синоним таблицы продажа абонементов;
* синоним таблицы информация о комплексах;
* синоним таблицы информация о записях посещения;
* синоним таблицы расписание;
* синоним таблицы список услуг;
* синоним таблицы типы тренировок;
* синоним таблицы список сотрудников;

**3.7. Вывод**

По итогу данного раздела были спроектированы и реализованы основные объекты базы данных, такие как:

* функции;
* процедуры;
* триггеры;
* материализованные и стандартные представления;
* синонимы;
* индексы;
* пакеты.

Все вышеперечисленные объекты базы данных обеспечивают удобное взаимодействие, а также являются предметами расширения возможностей в базе данных.

**4. Описание процедур импорта и экспорта**

Расширяемый язык разметки (XML) позволяет определять и хранить данные совместно используемым способом. XML поддерживает обмен информацией между компьютерными системами, такими как веб-сайты, базы данных и сторонние приложения. Предопределенные правила упрощают передачу данных в виде XML-файлов по любой сети, поскольку получатель может использовать эти правила для точного и эффективного чтения данных.

Процедуры импорта и экспорта являются важными инструментами для обмена данными между различными системами и приложениями. Импорт позволяет загружать данные из внешних источников в текущую систему, а экспорт - выгружать данные из текущей системы во внешние источники.

Для выполнения процедуры импорта необходимо определить источник данных, формат файла и метод передачи данных. Источник данных может быть представлен как локальным файлом, так и удаленным сервером. Формат файла может быть различным, например, CSV, XML, JSON и другие. Метод передачи данных может быть определен как однократный или периодический.

Для выполнения процедуры экспорта необходимо определить целевой источник данных, формат файла и метод передачи данных. Целевой источник данных может быть представлен как локальным файлом, так и удаленным сервером. Формат файла может быть различным, например, CSV, XML, JSON и другие. Метод передачи данных может быть определен как однократный или периодический.

Для обеспечения безопасности данных при выполнении процедур импорта и экспорта необходимо учитывать следующие меры:

* аутентификация и авторизация пользователей;
* шифрование данных при передаче;
* проверка целостности данных;
* ограничение доступа к данным только необходимым пользователям.

В зависимости от конкретных требований и условий, процедуры импорта и экспорта могут быть реализованы с помощью различных инструментов и технологий, таких как API, FTP, SFTP, HTTP и другие.

**4.1. Экспорт данных**

Данный метод экспорта данных в XML-формат использует пакет DBMS\_XMLGEN для генерации XML-документа на основе данных из таблицы AGENTE и функцию UTL\_FILE для записи сгенерированного XML-документа в файл.

Процедура начинается с определения переменных: l\_file - файловый дескриптор, xml\_data - строка, в которую будет записан сгенерированный XML-документ, и w\_repertoire - имя директории, в которую будет записан файл.

Затем происходит выполнение запроса на выборку данных из таблицы AGENTE с помощью функции dbms\_xmlgen.getxml, результат которого записывается в переменную xml\_data.

Далее определяется имя директории, в которую будет записан файл, и открывается файловый дескриптор с помощью функции UTL\_FILE.fopen.

Затем с помощью функции SYS.UTL\_FILE.putf происходит запись сгенерированного XML-документа в файл.

Наконец, файловый дескриптор закрывается с помощью функции SYS.UTL\_FILE.fclose.

На рисунке 4.1 представлен пример функции экспорта данных из таблиц в XML-файл.

CREATE OR REPLACE PROCEDURE EXPORT\_LIST\_OF\_CLIENTS\_TO\_XML

IS

    l\_file sys.utl\_file.file\_type;

    xml\_data VARCHAR2(32767);

    w\_repertoire VARCHAR2(64);

BEGIN

    SELECT dbms\_xmlgen.getxml('SELECT \* FROM LIST\_OF\_CLIENTS')

    AS xml INTO xml\_data FROM dual;

    w\_repertoire := 'DATA\_XML';

    l\_file := sys.utl\_file.fopen(w\_repertoire, 'LIST\_OF\_CLIENTS.xml', 'w');

    sys.utl\_file.putf(l\_file, '%s', xml\_data);

    sys.utl\_file.fclose(l\_file);

END;

Листинг 4.1 – Пример функции экспорта данных

Перевод данных в XML-файл имеет несколько важных преимуществ.

Во-первых, XML является универсальным форматом данных, который может быть использован для обмена данными между различными системами и приложениями. Это означает, что данные, сохраненные в XML-файле, могут быть легко переданы и обработаны другими системами, которые могут использовать различные языки программирования и платформы.

Во-вторых, XML обеспечивает структурирование данных, что делает их более понятными и удобными для обработки. XML-файлы могут содержать метаданные, которые описывают структуру данных, что позволяет легко определить, какие данные содержатся в файле и как они организованы.

В-третьих, XML-файлы могут быть легко обработаны с помощью различных инструментов и технологий, таких как XSLT, XPath и XQuery.

Эти инструменты позволяют извлекать, преобразовывать и анализировать данные в XML-файлах, что делает их более гибкими и удобными для использования.

В целом, перевод данных в XML-файл может упростить обмен данными между различными системами и приложениями, обеспечить структурирование данных и облегчить их обработку.

**4.2. Импорт данных**

Первая процедура, FILE\_TO\_CLOB, принимает имя файла в качестве входного параметра и возвращает CLOB (Character Large Object) в качестве выходного параметра. Она использует пакет UTL\_FILE для чтения содержимого XML-файла из каталога DATA\_XML. Затем она добавляет содержимое файла в выходной параметр CLOB до тех пор, пока не достигнет конца элемента ROWSET. Эта процедура может быть полезна, если вам нужно прочитать содержимое XML-файла и сохранить его в CLOB для дальнейшей обработки.

CREATE OR REPLACE PROCEDURE FILE\_TO\_CLOB(

p\_file\_name IN VARCHAR2,

p\_clob OUT CLOB

)

AS

v\_file UTL\_FILE.FILE\_TYPE;

v\_filename VARCHAR2(100);

v\_buffer VARCHAR2(32767);

BEGIN

-- Генерирование имени файла на основе имени таблицы

v\_filename := p\_file\_name || '.xml';

-- Чтение XML из файла

v\_file := UTL\_FILE.FOPEN('DATA\_XML', v\_filename, 'r');

LOOP

    UTL\_FILE.GET\_LINE(v\_file, v\_buffer);

    IF v\_buffer = '</ROWSET>' THEN

        p\_clob := p\_clob || v\_buffer;

        EXIT;

    ELSE

        p\_clob := p\_clob || v\_buffer;

    END IF;

END LOOP;

UTL\_FILE.FCLOSE(v\_file);

END;

/

Листинг 4.2 – Пример функции преобразования файла

Вторая процедура, IMPORT\_ABONEMENTS\_XML, принимает имя файла в качестве входного параметра и использует конструктор XMLTYPE для преобразования содержимого файла в объект XMLTYPE. Затем она извлекает значения элементов ID\_ABONEMENT, TYPE\_OF\_TRAINING, TYPE\_OF\_SERVICE, PRICE, AMOUNT\_OF\_VISITS, AMOUNT\_OF\_DAYS, AMOUNT\_OF\_MONTHS из каждого элемента ROW в XML и вставляет их в таблицу ABONEMENTS\_IMPORT. Если во время выполнения процедуры происходит ошибка, она выводит сообщение об ошибке с помощью функции DBMS\_OUTPUT.PUT\_LINE. Эта процедура может быть полезна, если вам нужно импортировать данные из XML-файла в таблицу базы данных. На рисунке 4.3 представлен пример функции импорта данных из таблиц в XML-файл.

CREATE OR REPLACE PROCEDURE IMPORT\_ABONEMENTS\_XML(

    p\_file IN VARCHAR2

)

AS

    v\_xml XMLTYPE := XMLTYPE(p\_file);

BEGIN

    FOR item IN (SELECT extractvalue(value(r), '/ROW/ID\_ABONEMENT')AS ID\_ABONEMENT,

                        extractvalue(value(r), '/ROW/TYPE\_OF\_TRAINING')AS TYPE\_OF\_TRAINING,

                        extractvalue(value(r), '/ROW/TYPE\_OF\_SERVICE')AS TYPE\_OF\_SERVICE,

                        extractvalue(value(r), '/ROW/PRICE')AS PRICE,

                        extractvalue(value(r), '/ROW/AMOUNT\_OF\_VISITS')AS AMOUNT\_OF\_VISITS,

                        extractvalue(value(r), '/ROW/AMOUNT\_OF\_DAYS')AS AMOUNT\_OF\_DAYS,

                        extractvalue(value(r), '/ROW/AMOUNT\_OF\_MONTHS')AS AMOUNT\_OF\_MONTHS FROM TABLE (XMLSEQUENCE(EXTRACT(v\_xml, '/ROWSET/ROW')))r

                        )

                        LOOP

                            INSERT INTO ABONEMENTS\_IMPORT VALUES(

                            TO\_NUMBER(item.ID\_ABONEMENT),

                            item.TYPE\_OF\_TRAINING,

                            item.TYPE\_OF\_SERVICE,

                            TO\_NUMBER(item.PRICE),

                            TO\_NUMBER(item.AMOUNT\_OF\_VISITS),

                            TO\_NUMBER(item.AMOUNT\_OF\_DAYS),

                            TO\_NUMBER(item.AMOUNT\_OF\_MONTHS)

                            );

                        END LOOP;

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' ||SQLERRM);

END;

Листинг 4.3 – Пример функции импорта данных

**4.3. Вывод**

Использование экспорта и импорта данных в XML-файлы в Oracle имеет множество преимуществ. Экспорт данных в XML-формат позволяет сохранить данные в универсальном формате, который может быть легко передан и обработан другими системами и приложениями. Кроме того, XML обеспечивает структурирование данных, что делает их более понятными и удобными для обработки. Импорт данных из XML-файлов в таблицы базы данных позволяет быстро и эффективно загружать большие объемы данных в базу данных. Кроме того, использование XML-файлов для импорта данных позволяет легко обрабатывать и анализировать данные с помощью различных инструментов и технологий, таких как XSLT, XPath и XQuery. В целом, использование экспорта и импорта данных в XML-файлы в Oracle может значительно упростить обмен данными между различными системами и приложениями, обеспечить структурирование данных и облегчить их обработку.

**5. Тестирование производительности**

Тестирование базы данных является важным этапом в разработке и поддержке приложений, которые используют базы данных. Это позволяет обнаружить и исправить ошибки и проблемы в базе данных, которые могут привести к неправильной работе приложения или потере данных.

Важность проведения тестирования базы данных заключается в следующем:

1. Обнаружение ошибок и проблем в базе данных. Тестирование базы данных позволяет выявить ошибки и проблемы, такие как неправильное хранение данных, некорректные связи между таблицами, неправильные индексы и другие проблемы, которые могут привести к неправильной работе приложения.

2. Повышение качества приложения. Тестирование базы данных помогает повысить качество приложения, так как обнаруженные ошибки и проблемы могут быть исправлены до того, как они приведут к серьезным проблемам.

3. Улучшение производительности. Тестирование базы данных может помочь улучшить производительность приложения, так как обнаруженные проблемы с индексами, запросами и другими аспектами базы данных могут быть исправлены.

4. Предотвращение потери данных. Тестирование базы данных помогает предотвратить потерю данных, так как обнаруженные проблемы с хранением данных могут быть исправлены до того, как они приведут к потере данных.

5. Соответствие требованиям безопасности. Тестирование базы данных помогает обеспечить соответствие требованиям безопасности, так как обнаруженные проблемы с безопасностью могут быть исправлены до того, как они приведут к утечке данных или другим проблемам.

В целом, проведение тестирования базы данных является важным этапом в разработке и поддержке приложений, которые используют базы данных. Это позволяет обнаружить и исправить ошибки и проблемы в базе данных, что повышает качество и производительность приложения и предотвращает потерю данных.

Тестирование производительности баз данных (БД) – это процесс оценки производительности БД в условиях реальной или смоделированной нагрузки. Цель тестирования производительности БД заключается в том, чтобы определить, как быстро и эффективно БД может обрабатывать запросы и обеспечивать доступ к данным при различных условиях нагрузки.

В процессе тестирования производительности БД могут использоваться различные инструменты и технологии, такие как симуляторы нагрузки, мониторинг производительности, анализаторы запросов и другие инструменты для сбора и анализа данных.

Результаты тестирования производительности БД могут быть использованы для оптимизации производительности БД, выявления узких мест и проблем в производительности, а также для принятия решений по масштабированию и оптимизации архитектуры БД.

Для проведения качественного тестирования разрабатываемой базы данных были добавлены, в сумме, 100000 строк в различные таблицы. Для добавления столь большого количества строк были разработаны различные анонимные блоки, которые заполняют таблицы рандомными данными, которые подходят по определению столбцов.

В качестве примера одного из таких анонимных блоков стоит взять анонимный блок для добавления 10000 строк в таблицу LIST\_OF\_CLIENTS.

DECLARE

    CHOICE NUMBER;

    POP NUMBER;

BEGIN

    FOR COUNTER IN 1..10000

    LOOP

        SELECT DBMS\_RANDOM.VALUE(18, 70) NUM INTO POP FROM DUAL;

        POP := ROUND(POP);

        SELECT DBMS\_RANDOM.VALUE(1, 10) NUM INTO CHOICE FROM DUAL;

        CHOICE := ROUND(CHOICE);

        IF(CHOICE > 5)

        THEN

            ADD\_ROWS\_TO\_LIST\_OF\_CLIENTS(FUNC\_SURNAME\_MEN, FUNC\_NAME\_MEN, FUNC\_PATRONYMIC\_MEN, POP, RAND\_ADDRESS, RAND\_PHONE\_NUMBER, RPASSPORT);

        ELSE

            ADD\_ROWS\_TO\_LIST\_OF\_CLIENTS(FUNC\_SURNAME\_WOMEN, FUNC\_NAME\_WOMEN, FUNC\_PATRONYMIC\_WOMEN, POP, RAND\_ADDRESS, RAND\_PHONE\_NUMBER, RPASSPORT);

        END IF;

    END LOOP;

    EXCEPTION

        WHEN OTHERS THEN

            DBMS\_OUTPUT.PUT\_LINE('ERROR: ' || SQLERRM);

            RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while adding rows to the LIST\_OF\_CLIENTS table.');

END;

Листинг 5.1 – Пример анонимного блока для заполнения данными

На рисунке 5.1 представлен пример анонимного блока для добавления 10000 строк в таблицу LIST\_OF\_CLIENTS.

Данный анонимный блок используется для добавления большого количества строк в таблицу LIST\_OF\_CLIENTS. Он начинается с определения переменных CHOICE и POP.

Затем происходит цикл FOR, который повторяется 10000 раз. В каждой итерации цикла происходит генерация случайных значений для переменных POP и CHOICE с помощью функции DBMS\_RANDOM.VALUE.

Далее происходит проверка значения переменной CHOICE. Если оно больше 5, то вызывается процедура INSERT\_ ADD\_ROWS\_TO\_LIST\_OF\_CLIENTS с параметрами FUNC\_SURNAME\_MEN, FUNC\_NAME\_MEN, FUNC\_PATRONYMIC\_MEN, POP, RAND\_ADDRESS, RAND\_PHONE\_NUMBER и RPASSPORT. Если значение CHOICE меньше или равно 5, то вызывается процедура INSERT\_ ADD\_ROWS\_TO\_LIST\_OF\_CLIENTS с параметрами FUNC\_SURNAME\_WOMEN, FUNC\_NAME\_WOMEN, FUNC\_PATRONYMIC\_WOMEN, POP, RAND\_ADDRESS, RAND\_PHONE\_NUMBER и RPASSPORT.

Для проведения тестирования также необходимы индексы по столбцам тестируемых таблиц.

На рисунке 5.2 представлен пример индекса для столбцов тестируемых таблиц.

CREATE INDEX INDEX\_SURNAME\_CLIENT ON LIST\_OF\_CLIENTS(SURNAME\_CLIENT);

Листинг 5.2 – Пример индекса для таблицы LIST\_OF\_CLIENTS

При выполнении запроса на количество всех строк таблицы LIST\_OF\_CLIENTS будет получен результат, представлен на рисунке 5.3.

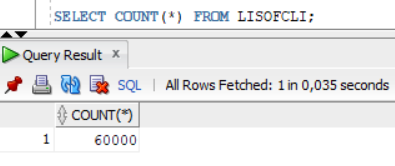


Рисунок 5.3 – Результат выполнения запроса количества всех строк таблицы LIST\_OF\_CLIENTS без использования индекса

Был получен результат 0,008 секунд, что является отличным результатом, однако при создании индекса для данной таблицы и повторения эксперимента тестирования будет получен результат, представленный на рисунке 5.4.

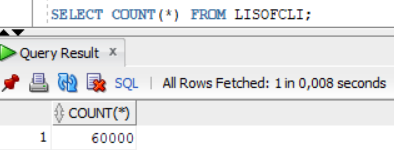


Рисунок 5.4 – Результат выполнения запроса количества всех строк таблицы LIST\_OF\_CLIENTS без использования индекса

Был получен результат 0,008 секунд, что свидетельствует об успешной работе индекса.

**5.3. Вывод**

Тестирование базы данных Oracle с использованием 100000 строк может быть очень полезным для определения производительности базы данных и ее способности обрабатывать большие объемы данных. Такое тестирование может помочь выявить возможные проблемы с производительностью, такие как медленные запросы или длительное время ответа, и позволить оптимизировать базу данных для более эффективной работы с большими объемами данных.

Одним из основных преимуществ тестирования базы данных с использованием 100000 строк является возможность проверить ее производительность в реальных условиях. Такое тестирование может помочь выявить проблемы с производительностью, которые могут возникнуть при работе с большими объемами данных, и позволить принять меры для их устранения.

Кроме того, тестирование базы данных с использованием 100000 строк может помочь определить ее способность обрабатывать большие объемы данных и поддерживать высокую производительность при работе с ними. Это может быть особенно важно для предприятий, которые работают с большими объемами данных и требуют высокой производительности базы данных для эффективной работы.

В целом, тестирование базы данных Oracle с использованием 100000 строк может быть очень полезным для определения ее производительности и способности обрабатывать большие объемы данных. Такое тестирование может помочь выявить проблемы с производительностью и оптимизировать базу данных для более эффективной работы с большими объемами данных.

**6. Описание технологии и ее применения в базе данных**

**6.1. Описание используемой технологии**

Технология мониторинга СУБД (системы управления базами данных) — это процесс наблюдения за работой базы данных с целью выявления проблем и улучшения ее производительности. Мониторинг СУБД включает в себя сбор и анализ различных метрик, таких как использование ресурсов, производительность запросов, нагрузка на сервер и другие параметры.

Применение технологии мониторинга СУБД в базе данных позволяет:

1. Определить проблемы производительности: мониторинг СУБД позволяет выявить проблемы производительности, такие как медленные запросы, блокировки и другие проблемы, которые могут замедлять работу базы данных.

2. Оптимизировать производительность: мониторинг СУБД позволяет оптимизировать производительность базы данных, путем выявления узких мест и оптимизации запросов и настройки параметров базы данных.

3. Предотвратить сбои: мониторинг СУБД позволяет выявлять проблемы до того, как они приведут к сбоям базы данных, что позволяет предотвратить потерю данных и снизить время простоя.

4. Улучшить безопасность: мониторинг СУБД позволяет выявлять потенциальные угрозы безопасности, такие как попытки несанкционированного доступа или взлома базы данных.

5. Улучшить планирование емкости: мониторинг СУБД позволяет определить, когда база данных достигнет предела своей емкости, что позволяет планировать расширение емкости заранее. Технология мониторинга СУБД может быть реализована с помощью различных инструментов, таких как SQL Profiler, Oracle Enterprise Manager, MySQL Workbench и другие. Она является важным инструментом для обеспечения эффективной и надежной работы базы данных. Дополнительно, технология мониторинга СУБД может быть использована для:

6. Определения требований к оборудованию: мониторинг СУБД позволяет определить требования к оборудованию, необходимому для обеспечения эффективной работы базы данных. Например, мониторинг может показать, что база данных нуждается в большем объеме оперативной памяти или процессорных ядер. 7. Определения проблем с сетью: мониторинг СУБД может помочь выявить проблемы с сетью, которые могут замедлять работу базы данных. Например, мониторинг может показать, что сеть перегружена или что существует проблема с соединением.

8. Определения проблем с приложениями: мониторинг СУБД может помочь выявить проблемы с приложениями, которые могут замедлять работу базы данных. Например, мониторинг может показать, что приложение отправляет слишком много запросов или что запросы не оптимизированы.

9. Определения проблем с базой данных: мониторинг СУБД может помочь выявить проблемы с самой базой данных, такие как повреждение данных или проблемы с индексами.

10. Определения требований к обслуживанию: мониторинг СУБД может помочь определить требования к обслуживанию базы данных, такие как резервное копирование, обновление программного обеспечения и другие задачи.

В целом, технология мониторинга СУБД является важным инструментом для обеспечения эффективной и надежной работы базы данных. Она позволяет выявлять проблемы и оптимизировать производительность базы данных, что в свою очередь повышает эффективность бизнес-процессов и улучшает качество обслуживания пользователей.

**6.3. Вывод**

Технология состояния мониторинга СУБД является важным инструментом для обеспечения надежности и эффективности работы баз данных. Она позволяет отслеживать различные параметры, такие как использование ресурсов, производительность и доступность, и предоставляет информацию о состоянии базы данных в режиме реального времени.

С помощью технологии мониторинга состояния СУБД можно быстро обнаруживать проблемы и устранять их до того, как они приведут к серьезным последствиям. Это позволяет повысить производительность и надежность баз данных, а также снизить риски потери данных.

Однако, необходимо учитывать, что технология мониторинга состояния СУБД требует определенных знаний и навыков для ее настройки и использования. Кроме того, она может потребовать дополнительных ресурсов для ее реализации, что может повлиять на общую производительность системы.

**Заключение**

По итогу выполнения заданий для курсового проекта была разработана база данных для оздоровительного центра. В процессе разработки базы данных была применена технология мониторинга состояния СУБД.

Для построения грамотой схемы базы данных были созданы и использованы объекты баз данных, такие как:

* триггеры;
* представления;
* материализованные представления;
* хранимые процедуры;
* функции;
* индексы;
* таблицы;
* пакеты;

Основной целью курсового проекта стало проектирование базы данных для дальнейшей интеграции с приложением, которое поможет облегчить взаимодействие с базой данных посредством программного интерфейса.

По итогу разработанного курсового проекта были реализованы следующие задачи:

* управление данными о комплексах (добавление и изменение сведений о комплексах)
* управление данными о клиентах (добавление и изменение дополнительной информации);
* управление данными о сотрудниках (добавление и изменение информации)
* анализ работы агентства (справки по количеству клиентов, заполненности туров в разные страны за различные периоды и в сравнении)

Было проведено грамотное тестирование базы данных, в рамках добавления 100000 строк и вызова различных процедур. Также были разработаны функции экспорта и импорта данных в XML.

Подводя итог по разработанному курсовому проекту, можно утверждать, что все необходимые требования были соблюдены и выполнены.

**Список используемых источников**

1. Язык C# и .NET Framework[Электронный ресурс] –http://professorweb.ru/my/csharp/charp\_theory/level1/infonet.php.

2. Руководство по ADO.NET и работе с базами данных [Электронный ресурс] –https://metanit.com/sharp/adonet/.

3. Документация Oracle[Электронный ресурс] / Foundation, Inc.https://docs.oracle.com/cd/B28359\_01/server.111/b31222/toc.htm

4. Официальный сайт Oracle[Электронный ресурс] / Foundation, Inc. <https://www.oracle.com/database/database-vault/index.html>

5. Продукты Oracle[Электронный ресурс] [Электронный ресурс] / Foundation, Inc. <http://www.interface.ru/home.asp?artId=24678>

6. Документы Oracle[Электронный ресурс] / Foundation, Inc. http://www.oracle.com/technetwork/database/security/database-vault-ds-12c-1898877.pdf

ПРИЛОЖЕНИЕ А Листинг скриптов создания таблиц

CREATE TABLE LIST\_OF\_CLIENTS

(

    ID\_CLIENT NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    SURNAME\_CLIENT NVARCHAR2(50) NOT NULL,

    NAME\_CLIENT NVARCHAR2(50) NOT NULL,

    PATRONYMIC\_CLIENT NVARCHAR2(50) NOT NULL,

    AGE\_CLIENT NUMBER NOT NULL,

    ADDRESS\_CLIENT NVARCHAR2(200) NOT NULL,

    TELEPHONE\_CLIENT NVARCHAR2(13) NOT NULL,

    PASSPORT\_CLIENT NVARCHAR2(20) NOT NULL,

    CONSTRAINT PK\_LIST\_OF\_CLIENTS PRIMARY KEY(ID\_CLIENT)

);

CREATE TABLE ABONEMENTS

(

    ID\_ABONEMENT NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    TYPE\_OF\_TRAINING NVARCHAR2(50) NOT NULL,

    TYPE\_OF\_SERVICE NVARCHAR2(50) NOT NULL,

    PRICE NUMBER NOT NULL,

    AMOUNT\_OF\_VISITS NUMBER NOT NULL,

    AMOUNT\_OF\_DAYS NUMBER NOT NULL,

    AMOUNT\_OF\_MONTHS NUMBER NOT NULL,

    CONSTRAINT PK\_ABONEMENTS PRIMARY KEY(ID\_ABONEMENT)

);

CREATE TABLE SALE\_OF\_ABONEMENTS

(

    ID\_SALE NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    ID\_CLIENT NUMBER NOT NULL,

    ID\_ABONEMENT NUMBER NOT NULL,

    DATA\_BEGIN DATE NOT NULL,

    DATA\_END DATE NOT NULL,

    CONSTRAINT PK\_SALE\_OF\_ABONEMENTS PRIMARY KEY(ID\_SALE),

    CONSTRAINT FK\_SALE\_OF\_ABONEMENTS FOREIGN KEY(ID\_CLIENT) REFERENCES LIST\_OF\_CLIENTS(ID\_CLIENT),

    CONSTRAINT FK\_SALE\_OF\_ABONEMENTS2 FOREIGN KEY(ID\_ABONEMENT) REFERENCES ABONEMENTS(ID\_ABONEMENT)

);

CREATE TABLE INFO\_COMPLEXES(

    ID\_COMPLEXES NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    NAME\_COMPLEXES NVARCHAR2(100),

    ADDRESS\_COMPLEXES NVARCHAR2(100),

    CAPACITY\_COMPLEXES NUMBER,

    CONSTRAINT PK\_INFO\_COMPLEXES PRIMARY KEY(ID\_COMPLEXES)

);

CREATE TABLE VISIT\_RECORD

(

    NUMBER\_OF\_VISITS NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    ID\_SALE NUMBER NOT NULL,

    NUMBER\_RECORD NUMBER NOT NULL,

    ID\_COMPLEXES NUMBER NOT NULL,

    CONSTRAINT PK\_VISIT\_RECORD PRIMARY KEY(NUMBER\_OF\_VISITS),

    CONSTRAINT FK\_VISIT\_RECORD FOREIGN KEY(ID\_SALE) REFERENCES SALE\_OF\_ABONEMENTS(ID\_SALE),

    CONSTRAINT FK\_VISIT\_RECORD2 FOREIGN KEY(NUMBER\_RECORD) REFERENCES TIME\_TABLE(NUMBER\_RECORD),

    CONSTRAINT FK\_VISIT\_RECORD3 FOREIGN KEY(ID\_COMPLEXES) REFERENCES INFO\_COMPLEXES(ID\_COMPLEXES)

);

CREATE TABLE TIME\_TABLE

(

    NUMBER\_RECORD NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    DATA\_TIME\_TABLE DATE NOT NULL,

    TIME\_BEGIN NVARCHAR2(20) NOT NULL,

    TIME\_END NVARCHAR2(20) NOT NULL,

    ID\_EMPLOYEE NUMBER NOT NULL,

    ID\_SERVICES NUMBER NOT NULL,

    ID\_TRAINING NUMBER NOT NULL,

    CONSTRAINT PK\_TIME\_TABLE PRIMARY KEY(NUMBER\_RECORD),

    CONSTRAINT FK\_TIME\_TABLE FOREIGN KEY(ID\_EMPLOYEE) REFERENCES LIST\_OF\_EMPLOYEE(ID\_EMPLOYEE),

    CONSTRAINT FK\_TIME\_TABLE2 FOREIGN KEY(ID\_SERVICES) REFERENCES LIST\_OF\_SERVICES(ID\_SERVICES),

    CONSTRAINT FK\_TIME\_TABLE3 FOREIGN KEY(ID\_TRAINING) REFERENCES TYPES\_OF\_TRAINING(ID\_TRAINING)

);

CREATE TABLE LIST\_OF\_SERVICES

(

    ID\_SERVICES NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    TYPES\_OF\_SERVICES NVARCHAR2(50) NOT NULL,

    CONSTRAINT PK\_LIST\_OF\_SERVICES PRIMARY KEY(ID\_SERVICES)

);

CREATE TABLE TYPES\_OF\_TRAINING

(

    ID\_TRAINING NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    TYPES\_OF\_TRAINING NVARCHAR2(50) NOT NULL,

    CONSTRAINT PK\_TYPES\_OF\_TRAINING PRIMARY KEY(ID\_TRAINING)

);

CREATE TABLE LIST\_OF\_EMPLOYEE

(

    ID\_EMPLOYEE NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    SURNAME\_EMPLOYEE NVARCHAR2(50) NOT NULL,

    NAME\_EMPLOYEE NVARCHAR2(50) NOT NULL,

    PATRONYMIC\_EMPLOYEE NVARCHAR2(50) NOT NULL,

    TELEPHONE\_EMPLOYEE NVARCHAR2(13) NOT NULL,

    ADDRESS\_EMPLOYEE NVARCHAR2(50) NOT NULL,

    SALARY\_EMPLOYEE NUMBER,

    CONSTRAINT PK\_LIST\_OF\_EMPLOYEE PRIMARY KEY(ID\_EMPLOYEE)

);

ID\_EMPLOYEE NUMBER NOT NULL,

LAST\_NAME nvarchar2(50) NOT NULL,

FIRST\_NAME nvarchar2(50) NOT NULL,

MIDDLE\_NAME nvarchar2(50) NOT NULL,

PHONE\_NUMBER\_EMPLOYEE nvarchar2(50) NOT NULL,

POST nvarchar2(50) NOT NULL,

CHIEF NUMBER,

CONSTRAINT PK\_ID\_EMPLOYEE

PRIMARY KEY (ID\_EMPLOYEE),

CONSTRAINT FK\_CHIEF

FOREIGN KEY (CHIEF)

REFERENCES TA\_EMPLOYEE(ID\_EMPLOYEE)

);

CREATE TABLE TA\_CLIENT

(

ID\_CLIENT NUMBER NOT NULL,

LAST\_NAME nvarchar2(50) NOT NULL,

FIRST\_NAME nvarchar2(50) NOT NULL,

MIDDLE\_NAME nvarchar2(50) NOT NULL,

ID\_EMPLOYEE NUMBER,

PHONE\_NUMBER\_CLIENT nvarchar2(13) NOT NULL,

PASSPORT nvarchar2(9) NOT NULl,

ADDRESS NUMBER NOT NULL,

CONSTRAINT PK\_ID\_CLIENT

PRIMARY KEY (ID\_CLIENT),

CONSTRAINT FK\_ID\_EMPLOYEE

FOREIGN KEY (ID\_EMPLOYEE)

REFERENCES TA\_EMPLOYEE(ID\_EMPLOYEE),

CONSTRAINT FKC\_ADDRESS

FOREIGN KEY (ADDRESS)

REFERENCES TA\_CLIENT\_ADDRESS(ID\_ADDRESS)

);

CREATE TABLE TA\_TOUR

(

ID\_TOUR NUMBER NOT NULL,

TITLE nvarchar2(50) NOT NULL,

PRICE NUMBER NOT NULL,

START\_DATE DATE NOT NULL,

END\_DATE DATE NOT NULL,

ID\_TOUR\_TYPE NUMBER NOT NULL,

ID\_TOUR\_COUNTRY NUMBER NOT NULL,

ID\_TOUR\_CITY NUMBER NOT NULL,

SEATS\_COUNT NUMBER NOT NULL,

IMAGE BLOB,

IMAGE\_LINK BFILE,

PICTURE ORDImage NOT NULL,

CONSTRAINT PK\_ID\_TOUR

PRIMARY KEY (ID\_TOUR),

CONSTRAINT FK\_ID\_TOUR\_TYPE

FOREIGN KEY (ID\_TOUR\_TYPE)

REFERENCES TA\_TOUR\_TYPE(ID\_TOUR\_TYPE)

);

CREATE TABLE TA\_SALE

(

ID\_SALE NUMBER NOT NULL,

ID\_CLIENT NUMBER NOT NULL,

ID\_EMPLOYEE NUMBER NOT NULL,

ID\_TOUR NUMBER NOT NULL,

SALE\_DATE DATE NOT NULL,

COUNT\_SEATS NUMBER NOT NULL,

CONSTRAINT PK\_ID\_SALE

PRIMARY KEY (ID\_SALE),

CONSTRAINT FKS\_ID\_CLIENT

FOREIGN KEY(ID\_CLIENT)

REFERENCES TA\_CLIENT(ID\_CLIENT),

CONSTRAINT FKS\_ID\_EMPLOYEE

FOREIGN KEY(ID\_EMPLOYEE)

REFERENCES TA\_EMPLOYEE(ID\_EMPLOYEE),

CONSTRAINT FKS\_ID\_TOUR

FOREIGN KEY(ID\_TOUR)

REFERENCES TA\_TOUR(ID\_TOUR)

);

CREATE TABLE TA\_REFUND

(

ID\_REFUND NUMBER NOT NULL,

ID\_CLIENT NUMBER NOT NULL,

ID\_EMPLOYEE NUMBER NOT NULL,

ID\_TOUR NUMBER NOT NULL,

REFUND\_DATE DATE NOT NULL,

COUNT\_SEATS NUMBER NOT NULL,

CONSTRAINT PK\_ID\_REFUND

PRIMARY KEY (ID\_REFUND),

CONSTRAINT FKF\_ID\_CLIENT

FOREIGN KEY(ID\_CLIENT)

REFERENCES TA\_CLIENT(ID\_CLIENT),

CONSTRAINT FKF\_ID\_EMPLOYEE

FOREIGN KEY(ID\_EMPLOYEE)

REFERENCES TA\_EMPLOYEE(ID\_EMPLOYEE),

CONSTRAINT FKF\_ID\_TOUR

FOREIGN KEY(ID\_TOUR)

REFERENCES TA\_TOUR(ID\_TOUR)

);

    CONSTRAINT PK\_INFO\_COMPLEXES PRIMARY KEY(ID\_COMPLEXES)

);

CREATE TABLE VISIT\_RECORD

(

    NUMBER\_OF\_VISITS NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    ID\_SALE NUMBER NOT NULL,

    NUMBER\_RECORD NUMBER NOT NULL,

    ID\_COMPLEXES NUMBER NOT NULL,

    CONSTRAINT PK\_VISIT\_RECORD PRIMARY KEY(NUMBER\_OF\_VISITS),

    CONSTRAINT FK\_VISIT\_RECORD FOREIGN KEY(ID\_SALE) REFERENCES SALE\_OF\_ABONEMENTS(ID\_SALE),

    CONSTRAINT FK\_VISIT\_RECORD2 FOREIGN KEY(NUMBER\_RECORD) REFERENCES TIME\_TABLE(NUMBER\_RECORD),

    CONSTRAINT FK\_VISIT\_RECORD3 FOREIGN KEY(ID\_COMPLEXES) REFERENCES INFO\_COMPLEXES(ID\_COMPLEXES)

);

CREATE TABLE TIME\_TABLE

(

    NUMBER\_RECORD NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    DATA\_TIME\_TABLE DATE NOT NULL,

    TIME\_BEGIN NVARCHAR2(20) NOT NULL,

    TIME\_END NVARCHAR2(20) NOT NULL,

    ID\_EMPLOYEE NUMBER NOT NULL,

    ID\_SERVICES NUMBER NOT NULL,

    ID\_TRAINING NUMBER NOT NULL,

    CONSTRAINT PK\_TIME\_TABLE PRIMARY KEY(NUMBER\_RECORD),

    CONSTRAINT FK\_TIME\_TABLE FOREIGN KEY(ID\_EMPLOYEE) REFERENCES LIST\_OF\_EMPLOYEE(ID\_EMPLOYEE),

    CONSTRAINT FK\_TIME\_TABLE2 FOREIGN KEY(ID\_SERVICES) REFERENCES LIST\_OF\_SERVICES(ID\_SERVICES),

    CONSTRAINT FK\_TIME\_TABLE3 FOREIGN KEY(ID\_TRAINING) REFERENCES TYPES\_OF\_TRAINING(ID\_TRAINING)

);

CREATE TABLE LIST\_OF\_SERVICES

(

    ID\_SERVICES NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    TYPES\_OF\_SERVICES NVARCHAR2(50) NOT NULL,

    CONSTRAINT PK\_LIST\_OF\_SERVICES PRIMARY KEY(ID\_SERVICES)

);

CREATE TABLE TYPES\_OF\_TRAINING

(

    ID\_TRAINING NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    TYPES\_OF\_TRAINING NVARCHAR2(50) NOT NULL,

    CONSTRAINT PK\_TYPES\_OF\_TRAINING PRIMARY KEY(ID\_TRAINING)

);

CREATE TABLE LIST\_OF\_EMPLOYEE

(

    ID\_EMPLOYEE NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    SURNAME\_EMPLOYEE NVARCHAR2(50) NOT NULL,

    NAME\_EMPLOYEE NVARCHAR2(50) NOT NULL,

    PATRONYMIC\_EMPLOYEE NVARCHAR2(50) NOT NULL,

    TELEPHONE\_EMPLOYEE NVARCHAR2(13) NOT NULL,

    ADDRESS\_EMPLOYEE NVARCHAR2(50) NOT NULL,

    SALARY\_EMPLOYEE NUMBER,

    CONSTRAINT PK\_LIST\_OF\_EMPLOYEE PRIMARY KEY(ID\_EMPLOYEE)

);

RENCES TA\_TOUR\_TYPE(ID\_TOUR\_TYPE)

);

CREATE TABLE TA\_SALE

(

ID\_SALE NUMBER NOT NULL,

ID\_CLIENT NUMBER NOT NULL,

ID\_EMPLOYEE NUMBER NOT NULL,

ID\_TOUR NUMBER NOT NULL,

SALE\_DATE DATE NOT NULL,

COUNT\_SEATS NUMBER NOT NULL,

CONSTRAINT PK\_ID\_SALE

PRIMARY KEY (ID\_SALE),

CONSTRAINT FKS\_ID\_CLIENT

FOREIGN KEY(ID\_CLIENT)

REFERENCES TA\_CLIENT(ID\_CLIENT),

CONSTRAINT FKS\_ID\_EMPLOYEE

FOREIGN KEY(ID\_EMPLOYEE)

REFERENCES TA\_EMPLOYEE(ID\_EMPLOYEE),

CONSTRAINT FKS\_ID\_TOUR

FOREIGN KEY(ID\_TOUR)

REFERENCES TA\_TOUR(ID\_TOUR)

);

CREATE TABLE TA\_REFUND

(

ID\_REFUND NUMBER NOT NULL,

ID\_CLIENT NUMBER NOT NULL,

ID\_EMPLOYEE NUMBER NOT NULL,

ID\_TOUR NUMBER NOT NULL,

REFUND\_DATE DATE NOT NULL,

COUNT\_SEATS NUMBER NOT NULL,

CONSTRAINT PK\_ID\_REFUND

PRIMARY KEY (ID\_REFUND),

CONSTRAINT FKF\_ID\_CLIENT

FOREIGN KEY(ID\_CLIENT)

REFERENCES TA\_CLIENT(ID\_CLIENT),

CONSTRAINT FKF\_ID\_EMPLOYEE

FOREIGN KEY(ID\_EMPLOYEE)

REFERENCES TA\_EMPLOYEE(ID\_EMPLOYEE),

CONSTRAINT FKF\_ID\_TOUR

FOREIGN KEY(ID\_TOUR)

REFERENCES TA\_TOUR(ID\_TOUR)

);

);

CREATE TABLE LIST\_OF\_EMPLOYEE

(

    ID\_EMPLOYEE NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1) NOT NULL,

    SURNAME\_EMPLOYEE NVARCHAR2(50) NOT NULL,

    NAME\_EMPLOYEE NVARCHAR2(50) NOT NULL,

    PATRONYMIC\_EMPLOYEE NVARCHAR2(50) NOT NULL,

    TELEPHONE\_EMPLOYEE NVARCHAR2(13) NOT NULL,

    ADDRESS\_EMPLOYEE NVARCHAR2(50) NOT NULL,

    SALARY\_EMPLOYEE NUMBER,

    CONSTRAINT PK\_LIST\_OF\_EMPLOYEE PRIMARY KEY(ID\_EMPLOYEE)

);

ПРИЛОЖЕНИЕ Б Листинг Процедуры добавления

CREATE OR REPLACE PROCEDURE ADD\_ROWS\_TO\_LIST\_OF\_CLIENTS(pSURNAME\_CLIENT NVARCHAR2, pNAME\_CLIENT NVARCHAR2, pPATRONYMIC\_CLIENT NVARCHAR2, pAGE\_CLIENT NUMBER, pADDRESS\_CLIENT NVARCHAR2, pTELEPHONE\_CLIENT NVARCHAR2, pPASSPORT\_CLIENT NVARCHAR2)

IS

BEGIN

INSERT INTO LISOFCLI(SURNAME\_CLIENT, NAME\_CLIENT, PATRONYMIC\_CLIENT, AGE\_CLIENT, ADDRESS\_CLIENT, TELEPHONE\_CLIENT, PASSPORT\_CLIENT)

VALUES(pSURNAME\_CLIENT, pNAME\_CLIENT, pPATRONYMIC\_CLIENT, pAGE\_CLIENT, pADDRESS\_CLIENT, pTELEPHONE\_CLIENT, pPASSPORT\_CLIENT);

EXCEPTION

WHEN OTHERS

THEN DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);

END;

DECLARE

CHOICE NUMBER;

POP NUMBER;

BEGIN

FOR COUNTER IN 1..40000

LOOP

SELECT DBMS\_RANDOM.VALUE(18, 70) NUM INTO POP FROM DUAL;

POP := ROUND(POP);

SELECT DBMS\_RANDOM.VALUE(1, 10) NUM INTO CHOICE FROM DUAL;

CHOICE := ROUND(CHOICE);

IF(CHOICE > 5)

THEN

ADD\_ROWS\_TO\_LIST\_OF\_CLIENTS(FUNC\_SURNAME\_MEN, FUNC\_NAME\_MEN, FUNC\_PATRONYMIC\_MEN, POP, RAND\_ADDRESS, RAND\_PHONE\_NUMBER, RPASSPORT);

ELSE

ADD\_ROWS\_TO\_LIST\_OF\_CLIENTS(FUNC\_SURNAME\_WOMEN, FUNC\_NAME\_WOMEN, FUNC\_PATRONYMIC\_WOMEN, POP, RAND\_ADDRESS, RAND\_PHONE\_NUMBER, RPASSPORT);

END IF;

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('ERROR: ' || SQLERRM);

RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while adding rows to the LIST\_OF\_CLIENTS table.');

END;

RAND\_PHONE\_NUMBER, RPASSPORT);

ELSE

ADD\_ROWS\_TO\_LIST\_OF\_CLIENTS(FUNC\_SURNAME\_WOMEN, FUNC\_NAME\_WOMEN, FUNC\_PATRONYMIC\_WOMEN, POP, RAND\_ADDRESS, RAND\_PHONE\_NUMBER, RPASSPORT);

END IF;

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('ERROR: ' || SQLERRM);

RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while adding rows to the LIST\_OF\_CLIENTS table.');

END;

CREATE OR REPLACE PROCEDURE ADD\_ROWS\_TO\_ABONEMENTS(pTYPE\_OF\_TRAINING NVARCHAR2, pTYPE\_OF\_SERVICE NVARCHAR2, pPRICE NUMBER, pAMOUNT\_OF\_VISITS NUMBER, pAMOUNT\_OF\_DAYS NUMBER,

pAMOUNT\_OF\_MONTHS NUMBER)

IS

BEGIN

INSERT INTO ABONM(TYPE\_OF\_TRAINING, TYPE\_OF\_SERVICE, PRICE, AMOUNT\_OF\_VISITS, AMOUNT\_OF\_DAYS, AMOUNT\_OF\_MONTHS)

VALUES(pTYPE\_OF\_TRAINING, pTYPE\_OF\_SERVICE, pPRICE, pAMOUNT\_OF\_VISITS, pAMOUNT\_OF\_DAYS, pAMOUNT\_OF\_MONTHS);

EXCEPTION

WHEN OTHERS

THEN DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);

END;

BEGIN

FOR COUNTER IN 1..40000

LOOP

ADD\_ROWS\_TO\_ABONEMENTS(RAND\_TYPE\_TRAINING, RAND\_TYPE\_SERVICES, RPRICE, AMOUNTVISIT, AMOUNTDAYS, AMOUNTMONTHS);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('ERROR: ' || SQLERRM);

RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while adding rows to the ABONEMENTS table.');

END;

CREATE OR REPLACE PROCEDURE ADD\_ROWS\_TO\_SALE\_OF\_ABONEMENTS(pIDCLIENT NUMBER, pIDABONEMENT NUMBER, pDATABEGIN DATE, pDATAEND DATE)

IS

BEGIN

INSERT INTO SALOFABONM(ID\_CLIENT, ID\_ABONEMENT, DATA\_BEGIN, DATA\_END)

VALUES (pIDCLIENT, pIDABONEMENT, pDATABEGIN, pDATAEND);

EXCEPTION

WHEN OTHERS

THEN DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);

END;

BEGIN

FOR COUNTER IN 1..40000

LOOP

ADD\_ROWS\_TO\_SALE\_OF\_ABONEMENTS(ID\_CLIENTS, ID\_ABONEMENTS, RTIME('01-05-2023', '31-05-2023'), RTIME('01-09-2023', '30-09-2023'));

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('ERROR: ' || SQLERRM);

RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while adding rows to the SALE\_OF\_ABONEMENTS table.');

END;

CREATE OR REPLACE PROCEDURE ADD\_ROWS\_TO\_INFO\_COMPLEXES(pNAME\_COMPLEXES NVARCHAR2, pADDRESS\_COMPLEXES NVARCHAR2, pCAPACITY\_COMPLEXES NUMBER)

IS

BEGIN

INSERT INTO INFCOMPL(NAME\_COMPLEXES, ADDRESS\_COMPLEXES, CAPACITY\_COMPLEXES)

VALUES(pNAME\_COMPLEXES, pADDRESS\_COMPLEXES, pCAPACITY\_COMPLEXES);

EXCEPTION

WHEN OTHERS

THEN DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);

END;

BEGIN

FOR COUNTER IN 1..100

LOOP

ADD\_ROWS\_TO\_INFO\_COMPLEXES(RAND\_NAME\_COMPLEX, RAND\_ADDRESS, CAPACITY\_C);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('ERROR: ' || SQLERRM);

RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while adding rows to the INFO\_COMPLEXES table.');

END;

CREATE OR REPLACE PROCEDURE ADD\_ROWS\_TO\_VISIT\_RECORD(pID\_SALE NUMBER, pNUMBER\_RECORD NUMBER, pCOMPLEXES\_ID NUMBER)

IS

BEGIN

INSERT INTO VISREC(ID\_SALE, NUMBER\_RECORD, ID\_COMPLEXES)

VALUES (pID\_SALE, pNUMBER\_RECORD, pCOMPLEXES\_ID);

EXCEPTION

WHEN OTHERS

THEN DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);

END;

BEGIN

FOR COUNTER IN 1..10

LOOP

ADD\_ROWS\_TO\_VISIT\_RECORD(ID\_SALE, NUMBER\_RECORD, ID\_COMPLEXES);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE|| 'ERROR: ' || SQLERRM);

RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while adding rows to the VISIT\_RECORD table.');

END;

CREATE OR REPLACE PROCEDURE ADD\_ROWS\_TO\_TIME\_TABLE(pDATA\_TIME\_TABLE DATE, pTIME\_BEGIN NVARCHAR2, pTIME\_END NVARCHAR2, pID\_EMPLOYEE NUMBER, pID\_SERVICES NUMBER, pID\_TRAINING NUMBER)

IS

BEGIN

INSERT INTO TIMTAB(DATA\_TIME\_TABLE, TIME\_BEGIN, TIME\_END, ID\_EMPLOYEE, ID\_SERVICES, ID\_TRAINING)

VALUES (pDATA\_TIME\_TABLE, pTIME\_BEGIN, pTIME\_END, pID\_EMPLOYEE, pID\_SERVICES, pID\_TRAINING);

EXCEPTION

WHEN OTHERS

THEN DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);

END;

BEGIN

FOR COUNTER IN 1..100

LOOP

ADD\_ROWS\_TO\_TIME\_TABLE(RTIME('01-06-2023', '31-08-2023'), RAND\_TIME\_BEGIN, RAND\_TIME\_END, ID\_EMPLOYEE, ID\_SERVICES, ID\_TRAINING);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('ERROR: ' || SQLERRM);

RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while adding rows to the TIME\_TABLE table.');

END;

CREATE OR REPLACE PROCEDURE ADD\_ROWS\_TO\_LIST\_OF\_SERVICES(pTYPES\_OF\_SERVICES NVARCHAR2)

IS

BEGIN

INSERT INTO LISOFSERV(TYPES\_OF\_SERVICES)

VALUES(pTYPES\_OF\_SERVICES);

EXCEPTION

WHEN OTHERS

THEN DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);

END;

BEGIN

FOR COUNTER IN 1..100

LOOP

ADD\_ROWS\_TO\_LIST\_OF\_SERVICES(RAND\_TYPE\_SERVICES);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('ERROR: ' || SQLERRM);

RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while adding rows to the LIST\_OF\_SERVICES table.');

END;

CREATE OR REPLACE PROCEDURE ADD\_ROWS\_TO\_TYPES\_OF\_TRAINING(pTYPES\_OF\_TRAINING NVARCHAR2)

IS

BEGIN

INSERT INTO TYPOFTRA(TYPES\_OF\_TRAINING)

VALUES(pTYPES\_OF\_TRAINING);

EXCEPTION

WHEN OTHERS

THEN DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);

END;

BEGIN

FOR COUNTER IN 1..100

LOOP

ADD\_ROWS\_TO\_TYPES\_OF\_TRAINING(RAND\_TYPE\_TRAINING);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('ERROR: ' || SQLERRM);

RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while adding rows to the TYPES\_OF\_TRAINING table.');

END;

CREATE OR REPLACE PROCEDURE ADD\_ROWS\_TO\_LIST\_OF\_EMPLOYEE(pSURNAME\_EMPLOYEE NVARCHAR2, pNAME\_EMPLOYEE NVARCHAR2, pPATRONYMIC\_EMPLOYEE NVARCHAR2, pTELEPHONE\_EMPLOYEE NVARCHAR2, pADDRESS\_EMPLOYEE NVARCHAR2, pSALARY\_EMPLOYEE NUMBER)

IS

BEGIN

INSERT INTO LISOFEMP(SURNAME\_EMPLOYEE, NAME\_EMPLOYEE, PATRONYMIC\_EMPLOYEE, TELEPHONE\_EMPLOYEE, ADDRESS\_EMPLOYEE, SALARY\_EMPLOYEE)

VALUES(pSURNAME\_EMPLOYEE, pNAME\_EMPLOYEE, pPATRONYMIC\_EMPLOYEE, pTELEPHONE\_EMPLOYEE, pADDRESS\_EMPLOYEE, pSALARY\_EMPLOYEE);

EXCEPTION

WHEN OTHERS

THEN DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);

END;

DECLARE

CHOICE NUMBER;

BEGIN

FOR COUNTER IN 1..100

LOOP

SELECT DBMS\_RANDOM.VALUE(1, 10) NUM INTO CHOICE FROM DUAL;

CHOICE := ROUND(CHOICE);

IF(CHOICE > 5)

THEN

ADD\_ROWS\_TO\_LIST\_OF\_EMPLOYEE(FUNC\_SURNAME\_MEN, FUNC\_NAME\_MEN, FUNC\_PATRONYMIC\_MEN, RAND\_PHONE\_NUMBER, RAND\_ADDRESS, RSALARY);

ELSE

ADD\_ROWS\_TO\_LIST\_OF\_EMPLOYEE(FUNC\_SURNAME\_MEN, FUNC\_NAME\_MEN, FUNC\_PATRONYMIC\_MEN, RAND\_PHONE\_NUMBER, RAND\_ADDRESS, RSALARY);

END IF;

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('ERROR: ' || SQLERRM);

RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while adding rows to the LIST\_OF\_EMPLOYEE table.');

END;

PATRONYMIC\_EMPLOYEE, TELEPHONE\_EMPLOYEE, ADDRESS\_EMPLOYEE, SALARY\_EMPLOYEE)

VALUES(pSURNAME\_EMPLOYEE, pNAME\_EMPLOYEE, pPATRONYMIC\_EMPLOYEE, pTELEPHONE\_EMPLOYEE, pADDRESS\_EMPLOYEE, pSALARY\_EMPLOYEE);

EXCEPTION

WHEN OTHERS

THEN DBMS\_OUTPUT.PUT\_LINE('CODE ERROR: '||SQLCODE||' MESSAGE: '||SQLERRM);

END;

DECLARE

CHOICE NUMBER;

BEGIN

FOR COUNTER IN 1..100

LOOP

SELECT DBMS\_RANDOM.VALUE(1, 10) NUM INTO CHOICE FROM DUAL;

CHOICE := ROUND(CHOICE);

IF(CHOICE > 5)

THEN

ADD\_ROWS\_TO\_LIST\_OF\_EMPLOYEE(FUNC\_SURNAME\_MEN, FUNC\_NAME\_MEN, FUNC\_PATRONYMIC\_MEN, RAND\_PHONE\_NUMBER, RAND\_ADDRESS, RSALARY);

ELSE

ADD\_ROWS\_TO\_LIST\_OF\_EMPLOYEE(FUNC\_SURNAME\_MEN, FUNC\_NAME\_MEN, FUNC\_PATRONYMIC\_MEN, RAND\_PHONE\_NUMBER, RAND\_ADDRESS, RSALARY);

END IF;

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('ERROR: ' || SQLERRM);

RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while adding rows to the LIST\_OF\_EMPLOYEE table.');

END;

ПРИЛОЖЕНИЕ В Листинг Export всех таблиц

CREATE OR REPLACE PROCEDURE EXPORT\_LIST\_OF\_CLIENTS\_TO\_XML

IS

l\_file sys.utl\_file.file\_type;

xml\_data VARCHAR2(32767);

w\_repertoire VARCHAR2(64);

BEGIN

SELECT dbms\_xmlgen.getxml('SELECT \* FROM LIST\_OF\_CLIENTS')

AS xml INTO xml\_data FROM dual;

w\_repertoire := 'DATA\_XML';

l\_file := sys.utl\_file.fopen(w\_repertoire, 'LIST\_OF\_CLIENTS.xml', 'w');

sys.utl\_file.putf(l\_file, '%s', xml\_data);

sys.utl\_file.fclose(l\_file);

END;

EXEC EXPORT\_LIST\_OF\_CLIENTS\_TO\_XML;

CREATE OR REPLACE PROCEDURE EXPORT\_ABONEMENTS\_TO\_XML

IS

l\_file sys.utl\_file.file\_type;

xml\_data VARCHAR2(32767);

w\_repertoire VARCHAR2(64);

BEGIN

SELECT dbms\_xmlgen.getxml('SELECT \* FROM ABONEMENTS')

AS xml INTO xml\_data FROM dual;

w\_repertoire := 'DATA\_XML';

l\_file := sys.utl\_file.fopen(w\_repertoire, 'ABONEMENTS.xml', 'w');

sys.utl\_file.putf(l\_file, '%s', xml\_data);

sys.utl\_file.fclose(l\_file);

END;

EXEC EXPORT\_ABONEMENTS\_TO\_XML;

CREATE OR REPLACE PROCEDURE EXPORT\_SALE\_OF\_ABONEMENTS\_XML

IS

l\_file sys.utl\_file.file\_type;

xml\_data VARCHAR2(32767);

w\_repertoire VARCHAR2(64);

BEGIN

SELECT dbms\_xmlgen.getxml('SELECT \* FROM SALE\_OF\_ABONEMENTS')

AS xml INTO xml\_data FROM dual;

w\_repertoire := 'DATA\_XML';

l\_file := sys.utl\_file.fopen(w\_repertoire, 'SALE\_OF\_ABONEMENTS.xml', 'w');

sys.utl\_file.putf(l\_file, '%s', xml\_data);

sys.utl\_file.fclose(l\_file);

END;

EXEC EXPORT\_SALE\_OF\_ABONEMENTS\_XML;

EXEC EXPORT\_SALE\_OF\_ABONEMENTS\_XML;

CREATE OR REPLACE PROCEDURE EXPORT\_INFO\_COMPLEXES\_XML

IS

l\_file sys.utl\_file.file\_type;

xml\_data VARCHAR2(32767);

w\_repertoire VARCHAR2(64);

BEGIN

SELECT dbms\_xmlgen.getxml('SELECT \* FROM INFO\_COMPLEXES')

AS xml INTO xml\_data FROM dual;

w\_repertoire := 'DATA\_XML';

l\_file := sys.utl\_file.fopen(w\_repertoire, 'INFO\_COMPLEXES.xml', 'w');

sys.utl\_file.putf(l\_file, '%s', xml\_data);

sys.utl\_file.fclose(l\_file);

END;

EXEC EXPORT\_INFO\_COMPLEXES\_XML;

CREATE OR REPLACE PROCEDURE EXPORT\_TIME\_TABLE\_XML

IS

l\_file sys.utl\_file.file\_type;

xml\_data VARCHAR2(32767);

w\_repertoire VARCHAR2(64);

BEGIN

SELECT dbms\_xmlgen.getxml('SELECT \* FROM TIME\_TABLE')

AS xml INTO xml\_data FROM dual;

w\_repertoire := 'DATA\_XML';

l\_file := sys.utl\_file.fopen(w\_repertoire, 'TIME\_TABLE.xml', 'w');

sys.utl\_file.putf(l\_file, '%s', xml\_data);

sys.utl\_file.fclose(l\_file);

END;

EXEC EXPORT\_TIME\_TABLE\_XML;

CREATE OR REPLACE PROCEDURE EXPORT\_LIST\_OF\_SERVICES\_XML

IS

l\_file sys.utl\_file.file\_type;

xml\_data VARCHAR2(32767);

w\_repertoire VARCHAR2(64);

BEGIN

SELECT dbms\_xmlgen.getxml('SELECT \* FROM LIST\_OF\_SERVICES')

AS xml INTO xml\_data FROM dual;

w\_repertoire := 'DATA\_XML';

l\_file := sys.utl\_file.fopen(w\_repertoire, 'LIST\_OF\_SERVICES.xml', 'w');

sys.utl\_file.putf(l\_file, '%s', xml\_data);

sys.utl\_file.fclose(l\_file);

END;

EXEC EXPORT\_LIST\_OF\_SERVICES\_XML;

CREATE OR REPLACE PROCEDURE EXPORT\_TYPES\_OF\_TRAINING\_XML

IS

l\_file sys.utl\_file.file\_type;

xml\_data VARCHAR2(32767);

w\_repertoire VARCHAR2(64);

BEGIN

SELECT dbms\_xmlgen.getxml('SELECT \* FROM TYPES\_OF\_TRAINING')

AS xml INTO xml\_data FROM dual;

w\_repertoire := 'DATA\_XML';

l\_file := sys.utl\_file.fopen(w\_repertoire, 'TYPES\_OF\_TRAINING.xml', 'w');

sys.utl\_file.putf(l\_file, '%s', xml\_data);

sys.utl\_file.fclose(l\_file);

END;

EXEC EXPORT\_TYPES\_OF\_TRAINING\_XML;

CREATE OR REPLACE PROCEDURE EXPORT\_LIST\_OF\_EMPLOYEE\_XML

IS

l\_file sys.utl\_file.file\_type;

xml\_data VARCHAR2(32767);

w\_repertoire VARCHAR2(64);

BEGIN

SELECT dbms\_xmlgen.getxml('SELECT \* FROM LIST\_OF\_EMPLOYEE')

AS xml INTO xml\_data FROM dual;

w\_repertoire := 'DATA\_XML';

l\_file := sys.utl\_file.fopen(w\_repertoire, 'LIST\_OF\_EMPLOYEE.xml', 'w');

sys.utl\_file.putf(l\_file, '%s', xml\_data);

sys.utl\_file.fclose(l\_file);

END;

EXEC EXPORT\_LIST\_OF\_EMPLOYEE\_XML;

ПРИЛОЖЕНИЕ Г Листинг Import всех таблиц

CREATE OR REPLACE PROCEDURE IMPORT\_LIST\_OF\_CLIENTS\_XML(

p\_file IN VARCHAR2

)

AS

v\_xml XMLTYPE := XMLTYPE(p\_file);

BEGIN

FOR item IN (SELECT extractvalue(value(r), '/ROW/ID\_CLIENT')AS ID\_CLIENT,

extractvalue(value(r), '/ROW/SURNAME\_CLIENT')AS SURNAME\_CLIENT,

extractvalue(value(r), '/ROW/NAME\_CLIENT')AS NAME\_CLIENT,

extractvalue(value(r), '/ROW/PATRONYMIC\_CLIENT')AS PATRONYMIC\_CLIENT,

extractvalue(value(r), '/ROW/AGE\_CLIENT')AS AGE\_CLIENT,

extractvalue(value(r), '/ROW/ADDRESS\_CLIENT')AS ADDRESS\_CLIENT,

extractvalue(value(r), '/ROW/TELEPHONE\_CLIENT')AS TELEPHONE\_CLIENT,

extractvalue(value(r), '/ROW/PASSPORT\_CLIENT')AS PASSPORT\_CLIENT FROM TABLE (XMLSEQUENCE(EXTRACT(v\_xml, '/ROWSET/ROW')))r

)

LOOP

INSERT INTO LIST\_OF\_CLIENTS\_IMPORT VALUES(

TO\_NUMBER(item.ID\_CLIENT),

item.SURNAME\_CLIENT,

item.NAME\_CLIENT,

item.PATRONYMIC\_CLIENT,

TO\_NUMBER(item.AGE\_CLIENT),

item.ADDRESS\_CLIENT,

item.TELEPHONE\_CLIENT,

item.PASSPORT\_CLIENT

);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' ||SQLERRM);

END;

DECLARE

v\_clob CLOB;

BEGIN

FILE\_TO\_CLOB('LIST\_OF\_CLIENTS', v\_clob);

IMPORT\_LIST\_OF\_CLIENTS\_XML(v\_clob);

COMMIT;

END;

CREATE OR REPLACE PROCEDURE IMPORT\_ABONEMENTS\_XML(

p\_file IN VARCHAR2

)

AS

v\_xml XMLTYPE := XMLTYPE(p\_file);

BEGIN

FOR item IN (SELECT extractvalue(value(r), '/ROW/ID\_ABONEMENT')AS ID\_ABONEMENT,

extractvalue(value(r), '/ROW/TYPE\_OF\_TRAINING')AS TYPE\_OF\_TRAINING,

extractvalue(value(r), '/ROW/TYPE\_OF\_SERVICE')AS TYPE\_OF\_SERVICE,

extractvalue(value(r), '/ROW/PRICE')AS PRICE,

extractvalue(value(r), '/ROW/AMOUNT\_OF\_VISITS')AS AMOUNT\_OF\_VISITS,

extractvalue(value(r), '/ROW/AMOUNT\_OF\_DAYS')AS AMOUNT\_OF\_DAYS,

extractvalue(value(r), '/ROW/AMOUNT\_OF\_MONTHS')AS AMOUNT\_OF\_MONTHS FROM TABLE (XMLSEQUENCE(EXTRACT(v\_xml, '/ROWSET/ROW')))r

)

LOOP

INSERT INTO ABONEMENTS\_IMPORT VALUES(

TO\_NUMBER(item.ID\_ABONEMENT),

item.TYPE\_OF\_TRAINING,

item.TYPE\_OF\_SERVICE,

TO\_NUMBER(item.PRICE),

TO\_NUMBER(item.AMOUNT\_OF\_VISITS),

TO\_NUMBER(item.AMOUNT\_OF\_DAYS),

TO\_NUMBER(item.AMOUNT\_OF\_MONTHS)

);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' ||SQLERRM);

END;

DECLARE

v\_clob CLOB;

BEGIN

FILE\_TO\_CLOB('ABONEMENTS', v\_clob);

IMPORT\_ABONEMENTS\_XML(v\_clob);

COMMIT;

END;

CREATE OR REPLACE PROCEDURE IMPORT\_SALE\_OF\_ABONEMENTS\_XML(

p\_file IN VARCHAR2

)

AS

v\_xml XMLTYPE := XMLTYPE(p\_file);

BEGIN

FOR item IN (SELECT extractvalue(value(r), '/ROW/ID\_SALE')AS ID\_SALE,

extractvalue(value(r), '/ROW/ID\_CLIENT')AS ID\_CLIENT,

extractvalue(value(r), '/ROW/ID\_ABONEMENT')AS ID\_ABONEMENT,

extractvalue(value(r), '/ROW/DATA\_BEGIN')AS DATA\_BEGIN,

extractvalue(value(r), '/ROW/DATA\_END')AS DATA\_END FROM TABLE (XMLSEQUENCE(EXTRACT(v\_xml, '/ROWSET/ROW')))r

)

LOOP

INSERT INTO SALE\_OF\_ABONEMENTS\_IMPORT VALUES(

TO\_NUMBER(item.ID\_SALE),

TO\_NUMBER(item.ID\_CLIENT),

TO\_NUMBER(item.ID\_ABONEMENT),

TO\_DATE(item.DATA\_BEGIN),

TO\_DATE(item.DATA\_END)

);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' ||SQLERRM);

END;

DECLARE

v\_clob CLOB;

BEGIN

FILE\_TO\_CLOB('SALE\_OF\_ABONEMENTS', v\_clob);

IMPORT\_SALE\_OF\_ABONEMENTS\_XML(v\_clob);

COMMIT;

END;

CREATE OR REPLACE PROCEDURE IMPORT\_INFO\_COMPLEXES\_XML(

p\_file IN VARCHAR2

)

AS

v\_xml XMLTYPE := XMLTYPE(p\_file);

BEGIN

FOR item IN (SELECT extractvalue(value(r), '/ROW/ID\_COMPLEXES')AS ID\_COMPLEXES,

extractvalue(value(r), '/ROW/NAME\_COMPLEXES')AS NAME\_COMPLEXES,

extractvalue(value(r), '/ROW/ADDRESS\_COMPLEXES')AS ADDRESS\_COMPLEXES,

extractvalue(value(r), '/ROW/CAPACITY\_COMPLEXES')AS CAPACITY\_COMPLEXES FROM TABLE

)

LOOP

INSERT INTO INFO\_COMPLEXES\_IMPORT VALUES(

TO\_NUMBER(item.ID\_COMPLEXES),

item.NAME\_COMPLEXES,

item.ADDRESS\_COMPLEXES,

TO\_NUMBER(item.CAPACITY\_COMPLEXES)

);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' ||SQLERRM);

END;

DECLARE

v\_clob CLOB;

BEGIN

FILE\_TO\_CLOB('INFO\_COMPLEXES', v\_clob);

IMPORT\_INFO\_COMPLEXES\_XML(v\_clob);

COMMIT;

END;

(XMLSEQUENCE(EXTRACT(v\_xml, '/ROWSET/ROW')))r

)

LOOP

INSERT INTO INFO\_COMPLEXES\_IMPORT VALUES(

TO\_NUMBER(item.ID\_COMPLEXES),

item.NAME\_COMPLEXES,

item.ADDRESS\_COMPLEXES,

TO\_NUMBER(item.CAPACITY\_COMPLEXES)

);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' ||SQLERRM);

END;

DECLARE

v\_clob CLOB;

BEGIN

FILE\_TO\_CLOB('INFO\_COMPLEXES', v\_clob);

IMPORT\_INFO\_COMPLEXES\_XML(v\_clob);

COMMIT;

END;

CREATE OR REPLACE PROCEDURE IMPORT\_TIME\_TABLE\_XML(

p\_file IN VARCHAR2

)

AS

v\_xml XMLTYPE := XMLTYPE(p\_file);

BEGIN

FOR item IN (SELECT extractvalue(value(r), '/ROW/NUMBER\_RECORD')AS NUMBER\_RECORD,

extractvalue(value(r), '/ROW/DATA\_TIME\_TABLE')AS DATA\_TIME\_TABLE,

extractvalue(value(r), '/ROW/TIME\_BEGIN')AS TIME\_BEGIN,

extractvalue(value(r), '/ROW/TIME\_END')AS TIME\_END,

extractvalue(value(r), '/ROW/ID\_EMPLOYEE')AS ID\_EMPLOYEE,

extractvalue(value(r), '/ROW/ID\_SERVICES')AS ID\_SERVICES,

extractvalue(value(r), '/ROW/ID\_TRAINING')AS ID\_TRAINING FROM TABLE (XMLSEQUENCE(EXTRACT(v\_xml, '/ROWSET/ROW')))r

)

LOOP

INSERT INTO TIME\_TABLE\_IMPORT VALUES(

TO\_NUMBER(item.NUMBER\_RECORD),

TO\_DATE(item.DATA\_TIME\_TABLE),

item.TIME\_BEGIN,

item.TIME\_END,

TO\_NUMBER(item.ID\_EMPLOYEE),

TO\_NUMBER(item.ID\_SERVICES),

TO\_NUMBER(item.ID\_TRAINING)

);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' ||SQLERRM);

END;

DECLARE

v\_clob CLOB;

BEGIN

FILE\_TO\_CLOB('TIME\_TABLE', v\_clob);

IMPORT\_TIME\_TABLE\_XML(v\_clob);

COMMIT;

END;

);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' ||SQLERRM);

END;

DECLARE

v\_clob CLOB;

BEGIN

FILE\_TO\_CLOB('TIME\_TABLE', v\_clob);

IMPORT\_TIME\_TABLE\_XML(v\_clob);

COMMIT;

END;

CREATE OR REPLACE PROCEDURE IMPORT\_LIST\_OF\_SERVICES\_XML(

p\_file IN VARCHAR2

)

AS

v\_xml XMLTYPE := XMLTYPE(p\_file);

BEGIN

FOR item IN (SELECT extractvalue(value(r), '/ROW/ID\_SERVICES')AS ID\_SERVICES,

extractvalue(value(r), '/ROW/TYPES\_OF\_SERVICES')AS TYPES\_OF\_SERVICES FROM TABLE (XMLSEQUENCE(EXTRACT(v\_xml, '/ROWSET/ROW')))r

)

LOOP

INSERT INTO LIST\_OF\_SERVICES\_IMPORT VALUES(

TO\_NUMBER(item.ID\_SERVICES),

item.TYPES\_OF\_SERVICES

);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' ||SQLERRM);

END;

DECLARE

v\_clob CLOB;

BEGIN

FILE\_TO\_CLOB('LIST\_OF\_SERVICES', v\_clob);

IMPORT\_LIST\_OF\_SERVICES\_XML(v\_clob);

COMMIT;

END;

CREATE OR REPLACE PROCEDURE IMPORT\_TYPES\_OF\_TRAINING\_XML(

p\_file IN VARCHAR2

)

AS

v\_xml XMLTYPE := XMLTYPE(p\_file);

BEGIN

FOR item IN (SELECT extractvalue(value(r), '/ROW/ID\_TRAINING')AS ID\_TRAINING,

extractvalue(value(r), '/ROW/TYPES\_OF\_TRAINING')AS TYPES\_OF\_TRAINING FROM TABLE (XMLSEQUENCE(EXTRACT(v\_xml, '/ROWSET/ROW')))r

)

LOOP

INSERT INTO TYPES\_OF\_TRAINING\_IMPORT VALUES(

TO\_NUMBER(item.ID\_TRAINING),

item.TYPES\_OF\_TRAINING

);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' ||SQLERRM);

END;

DECLARE

v\_clob CLOB;

BEGIN

FILE\_TO\_CLOB('TYPES\_OF\_TRAINING', v\_clob);

IMPORT\_TYPES\_OF\_TRAINING\_XML(v\_clob);

COMMIT;

END;

CREATE OR REPLACE PROCEDURE IMPORT\_LIST\_OF\_EMPLOYEE\_XML(

p\_file IN VARCHAR2

)

AS

v\_xml XMLTYPE := XMLTYPE(p\_file);

BEGIN

FOR item IN (SELECT extractvalue(value(r), '/ROW/ID\_EMPLOYEE')AS ID\_EMPLOYEE,

extractvalue(value(r), '/ROW/SURNAME\_EMPLOYEE')AS SURNAME\_EMPLOYEE,

extractvalue(value(r), '/ROW/NAME\_EMPLOYEE')AS NAME\_EMPLOYEE,

extractvalue(value(r), '/ROW/PATRONYMIC\_EMPLOYEE')AS PATRONYMIC\_EMPLOYEE,

extractvalue(value(r), '/ROW/TELEPHONE\_EMPLOYEE')AS TELEPHONE\_EMPLOYEE,

extractvalue(value(r), '/ROW/ADDRESS\_EMPLOYEE')AS ADDRESS\_EMPLOYEE,

extractvalue(value(r), '/ROW/SALARY\_EMPLOYEE')AS SALARY\_EMPLOYEE FROM TABLE (XMLSEQUENCE(EXTRACT(v\_xml, '/ROWSET/ROW')))r

)

LOOP

INSERT INTO LIST\_OF\_EMPLOYEE\_IMPORT VALUES(

TO\_NUMBER(item.ID\_EMPLOYEE),

item.SURNAME\_EMPLOYEE,

item.NAME\_EMPLOYEE,

item.PATRONYMIC\_EMPLOYEE,

item.TELEPHONE\_EMPLOYEE,

item.ADDRESS\_EMPLOYEE,

TO\_NUMBER(item.SALARY\_EMPLOYEE)

);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' ||SQLERRM);

END;

DECLARE

v\_clob CLOB;

BEGIN

FILE\_TO\_CLOB('LIST\_OF\_EMPLOYEE', v\_clob);

IMPORT\_LIST\_OF\_EMPLOYEE\_XML(v\_clob);

COMMIT;

END;

LOOP

INSERT INTO LIST\_OF\_EMPLOYEE\_IMPORT VALUES(

TO\_NUMBER(item.ID\_EMPLOYEE),

item.SURNAME\_EMPLOYEE,

item.NAME\_EMPLOYEE,

item.PATRONYMIC\_EMPLOYEE,

item.TELEPHONE\_EMPLOYEE,

item.ADDRESS\_EMPLOYEE,

TO\_NUMBER(item.SALARY\_EMPLOYEE)

);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' ||SQLERRM);

END;

DECLARE

v\_clob CLOB;

BEGIN

FILE\_TO\_CLOB('LIST\_OF\_EMPLOYEE', v\_clob);

IMPORT\_LIST\_OF\_EMPLOYEE\_XML(v\_clob);

COMMIT;

END;

ПРИЛОЖЕНИЕ Д Листинг Синонимы всех таблиц

CREATE PUBLIC SYNONYM LISOFCLI FOR KP\_OWNER.LIST\_OF\_CLIENTS;

CREATE PUBLIC SYNONYM ABONM FOR KP\_OWNER.ABONEMENTS;

CREATE PUBLIC SYNONYM SALOFABONM FOR KP\_OWNER.SALE\_OF\_ABONEMENTS;

CREATE PUBLIC SYNONYM INFCOMPL FOR KP\_OWNER.INFO\_COMPLEXES;

CREATE PUBLIC SYNONYM VISREC FOR KP\_OWNER.VISIT\_RECORD;

CREATE PUBLIC SYNONYM TIMTAB FOR KP\_OWNER.TIME\_TABLE;

CREATE PUBLIC SYNONYM LISOFSERV FOR KP\_OWNER.LIST\_OF\_SERVICES;

CREATE PUBLIC SYNONYM TYPOFTRA FOR KP\_OWNER.TYPES\_OF\_TRAINING;

CREATE PUBLIC SYNONYM LISOFEMP FOR KP\_OWNER.LIST\_OF\_EMPLOYEE;