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

Факультет Информационных технологий

Кафедра Информационных систем и технологий

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

**КУРСОВАЯ ЗАПИСКА**

**к курсовому проектированию**

**по дисциплине** "Базы данных"

**Исполнитель**

студент(а) 2 курса 3 группы А. Н. Халалеенко

подпись, дата

Курсовой проект защищен с оценкой

**Руководитель**

ассистент  Н. И. Уласевич

должность, учен. степень, ученое звание подпись, дата

Минск 2023

**Оглавление**

[Введение 5](#_Toc134897913)

[1. Постановка задачи 6](#_Toc134897914)

[1.1 Аналитический обзор аналога «Яндекс-доставка» 6](#_Toc134897915)

[1.2 Аналитический обзор аналога «Куфар-доставка» 6](#_Toc134897916)

[1.3 Технические средства, задача и цель 6](#_Toc134897917)

[1.4 Техгология 7](#_Toc134897918)

[1.5 Вывод по разделу 7](#_Toc134897919)

[2. Проектирование базы данных 8](#_Toc134897920)

[2.1 Разработка пользователей 8](#_Toc134897921)

[2.2 Разработка таблиц базы данных 11](#_Toc134897922)

[2.3 Диаграммы UML 13](#_Toc134897923)

[3. Разработка объектов базы данных 15](#_Toc134897924)

[3.1 Последовательности 15](#_Toc134897925)

[3.2 Представления 16](#_Toc134897926)

[3.3 Триггеры 18](#_Toc134897927)

[3.4 Индексы 20](#_Toc134897928)

[3.5 Материализованное представление 20](#_Toc134897929)

[3.6 Хранимые процедуры 22](#_Toc134897930)

[3.7 Функции 24](#_Toc134897931)

[3.8 Вывод по разделу 29](#_Toc134897932)

[4. Описание процедур импорта и экспорта 30](#_Toc134897933)

[4.1 Импорт данных 30](#_Toc134897934)

[4.2 Экспорт данных 31](#_Toc134897935)

[5. Тестирование производительности 32](#_Toc134897936)

[5.1 Заполнение таблиц данными 32](#_Toc134897937)

[5.2 Создание дополнительных индексов 32](#_Toc134897938)

[6. Описание технологии и её применения в базе данных 34](#_Toc134897939)

[7. Краткое описание приложения для демонстрации 37](#_Toc134897940)

[8. Руководство пользователя 40](#_Toc134897941)

[Заключение 41](#_Toc134897942)

[Список используемых источников 42](#_Toc134897943)

[Приложение А Листинг кода для создания таблиц 43](#_Toc134897944)

[Приложение Б Листинг кода процедур для работы с базой данных 45](#_Toc134897945)

# **Введение**

С тех пор, как человек стал думать о будущем и более тщательно учитывать настоящее, ему понадобилась помощь в анализе и учете имеющихся данных, т.к. ему перестало хватать собственной памяти для того, чтобы учитывать все, что происходит вокруг или того, что у него имеется. Со времен появления письменности, это стало делать проще.

С 1955г., когда появилось программируемое оборудование обработки записей, начинается история хранилищ данных в узком смысле, в том, каким мы теперь его знаем. В соответствии с определением, введенным Кристофером Дейтом в его классическом учебнике «Введение в системы баз данных», база данных - это организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей. Как бы то ни было, отличительной особенностью современных баз данных является то, что они хранятся и обрабатываются в вычислительной системе.

Для данного курсового проекта была выбрана такая тема, как «Реализация базы данных службы доставки с использованием технологии Apex». Данная тема является более чем актуальной, т.к. в современном мире услугами доставки пользуется довольно большая часть населения крупных городов, т.к. имея большую площадь города, людям трудно передвигаться по нему. Не всегда общественный транспорт бывает удобен, постоянные пробки, час пики – всё это затрудняет передвижение населения по городам, мегаполисам и агломерациям. Как раз для таких ситуаций и были разработаны сервисы, осуществляющие доставку различных товаров. Не нужно куда-то выходить, передвигаться или ехать. Достаточно просто воспользоваться приложением, позволяющим всего за пару минут оформить доставку необходимого товара.

Для пользователя всё должно выглядеть наилучшим образом – удобство пользования приложением, быстрота и отзывчивость службы поддержки в случае нестандартных ситуаций. Но, вот, компаниям, владеющим такими сервисами, нужно как-то вести учёт своих работ, для того чтобы, например, отчитываться пред налоговой, понимать масштаб работ и учитывать тенденции роста или падений спроса на предлагаемые ими услуги. Разработав план и учтя все риски и недостатки, новые игроки на рынке начинают свою деятельность и им обязательно нужна база данных, способная обрабатывать запросы необходимых масштабов. В данном курсовом проекте, в качестве сервера базы данных, был выбран Oracle. Однако, в качестве технологии, будет использоваться Oracle Apex.

Данная тема будет актуальна ещё долгое время, т.к. такой модели разработки сервисов и приложений придерживаются многие крупные игроки на рынке. Следовательно, выполненная работа может послужить хорошим примером для разработчиков, начинающих движение в этом направлении.

# **Постановка задачи**

В качестве аналогов продукту, который разрабатывается в рамках текущий курсовой работы, можно выделить два основных конкурента, работающих на территории, конкурирующей с нашим продуктом: Яндекс-доставка и Куфар-доставка. Оба сервиса работают одинаковым способом – через приложение.

## **Аналитический обзор аналога «Яндекс-доставка»**

Яндекс-доставка - это часть экосистемы Яндекс, которая предлагает широкий спектр услуг доставки, включая курьерскую доставку, доставку из магазинов и ресторанов, а также доставку товаров от частных продавцов. Одним из главных преимуществ Яндекс-доставки является ее широкий выбор услуг доставки.

Кроме того, Яндекс-доставка предлагает удобный интерфейс, который позволяет пользователям легко заказывать и отслеживать доставку своих товаров. Сервис также предоставляет удобную систему оплаты, которая позволяет оплачивать заказы наличными или банковской картой.

## **Аналитический обзор аналога «Куфар-доставка»**

Куфар-доставка также предоставляет удобные услуги доставки для покупателей. Она обеспечивает доставку товаров в более чем 1200 городах России, а также предлагает удобный инструмент для отслеживания статуса доставки. Куфар-доставка также предлагает своим клиентам гибкую систему оплаты, включая оплату наличными или банковской картой.

В целом, Яндекс-доставка и Куфар-доставка предлагают удобные услуги доставки товаров в России. Выбор между ними будет зависеть от индивидуальных потребностей и предпочтений клиента.

## **Технические средства, задача и цель**

В качестве технических средств, которые должны использоваться машиной, выступающей в роли сервера, отметим следующий необходимый минимум: обязательное наличие нескольких запоминающих устройств для возможности хранения нескольких экземпляром основных файлов базы данных, таких как контрольные файлы, файлы параметров, файлы паролей и прочие, архивирования существующих данных с возможностью технологии backup; мощного центрального процессора, способного обрабатывать большое количество операций в секунду; устройства ввода-вывода для возможности администрирования базы данных прямо с сервера; любой перечень необходимого оборудования, организующего выход данной машины в сеть интернет.

Цель устанавливается следующая – разработать сервис доставки, который должен содержать базу данных с хранимой в ней информацией и возможностью управления её.

Задача – создание базы данных и приложения для представления информации в ней.

## **Технология**

В качестве технологии будет использоваться Oracle Apex. Как заверяют сами разработчики, Oracle Apex - это полностью управляемая платформа разработки приложений с минимальным программированием для создания и развертывания современных приложений, работающих на основе данных, в Oracle Cloud. Бизнес-пользователи и разработчики приложений могут создавать корпоративные приложения в 38 раз быстрее, чем при написании кода вручную, не тратя при этом сил на изучение сложных веб-технологий. Коротко говоря, Apex позволяет быстро разрабатывать веб-приложения любых масштабов с базами данных, делая это быстро и не затрудняясь, за счёт удобного и понятного интерфейса. Можно не беспокоиться за хранение своих данных – всё находится в облаке Oracle Cloud; достаточно лишь запомнить свои учётные данные при создании приложения.

## **Вывод по разделу**

По итогу говоря, при разработке приложения были учтены все требования, придерживаясь которых, можно создать современное и конкурентноспособное приложение.

# **Проектирование базы данных**

## **Разработка пользователей**

Для начала, стоит сказать, что любое приложение имеет свою систему иерархий и привилегий. В данном случае, речь идёт о пользователях нашего приложения и их доступе к функциям и возможностям нашего продукта. Для данного сервиса мы создадим, для начала, пятерых основных пользователей со следующими ролями: главный администратор рабочего пространства, разработчик объектов базы данных, имеющий все права разработчика, разработчик приложения, разработчик инфраструктуры базы данных, имеющий права лишь на sql-разработку и пользователь. Разработчики бывают разные, с разными привилегиями и уровнями доступа к возможностям приложения и базы данных. Именно такие и будут созданы.

Важно уточнить то, что необходимо грамотно продумывать момент создания пользователей и ролей приложения. Ведь, если сотрудник с определённой ролью будет иметь доступ к данным, которые ему не должны быть видны, то база данных может потерять свою ценную информацию, либо поделится. Также, стоит грамотно выдавать привилегии разработчикам, чтобы они не смогли сломать сервис, то есть какую-то часть приложения, базы данных, либо, что-нибудь ещё.

Приступим. Для создания пользователей с привилегиями воспользуемся встроенными средствами Apex.

Для начала, создадим администратора нашего рабочего пространства. То есть, этой роли доступны абсолютно любые действия. Окно создания администратора представлено на рисунке 2.1.

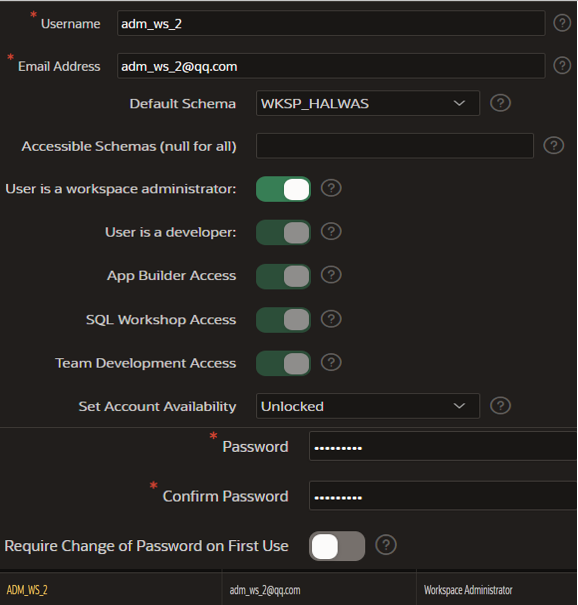


Рисунок 2.1 – Создание пользователя adm\_ws\_2

Как видно на рисунке, пользователь adm\_ws\_2 обладает такими правами.

Далее, создадим пользователя, имеющего права на полноценную разработку приложения и инфраструктуры базы данных. Данный пользователь сможет создавать в нашем сервисе все объекты, которые пожелает. То есть, ему будут доступны как объекты базы данных, так и объекты визуальной части приложения, реализующего взаимодействие пользователей с сервисом. Этому разработчику будет доступна командная разработка. То есть, он сможет разрабатывать сервис совместно с другими разработчиками. Создание такого пользователя представлено на рисунке 2.2.

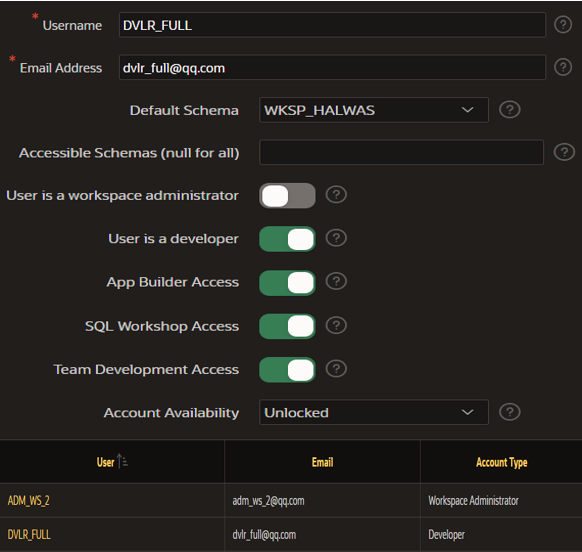


Рисунок 2.2 – Создание пользователя dvlr\_full

Пользователь dvlr\_full обладает всеми правами, кроме администрирования.

Приступим к созданию разработчика, который будет заниматься разработкой объектов нашей базы данных, что продемонстрировано на рисунке 2.3.

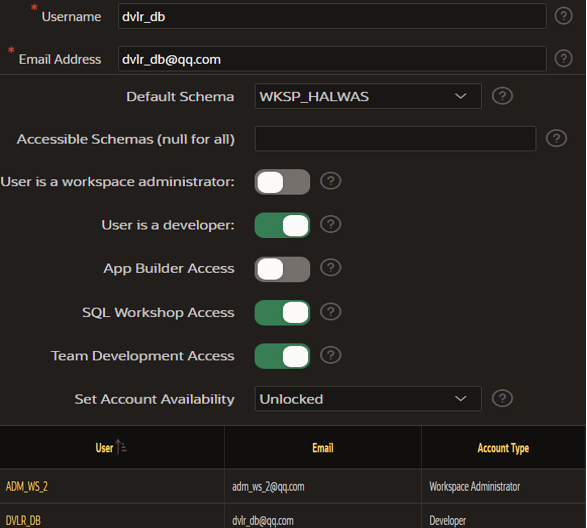


Рисунок 2.3 – Создание разработчика объектов базы данных

Пользователю dvlr\_db будет не лишним доступ к совместной разработке.

Таким же образом, создадим разработчика, имеющим доступ к разработке приложения. Демонстрация на рисунке 2.4.

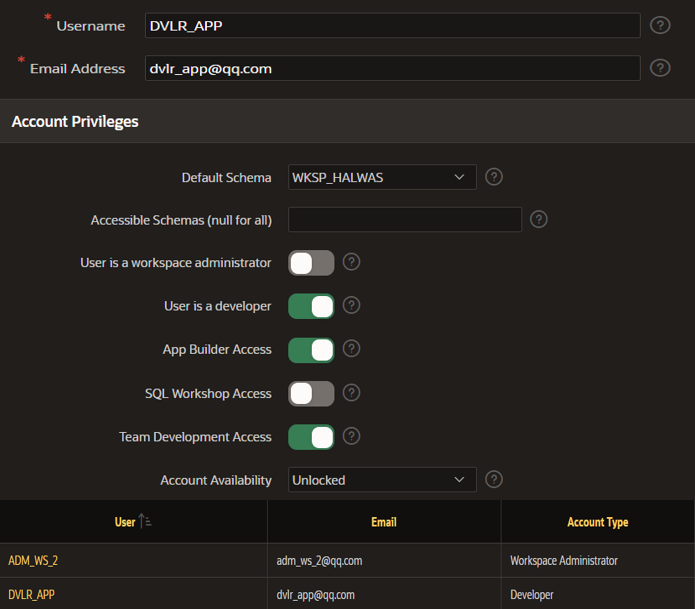


Рисунок 2.4 – Создание разработчика dvlr\_app

Пользователю dvlr\_app также доступна совместная разработка.

Что нам осталось на данном – это создать пользователя нашего приложения. Следует отметить, что данная роль предназначена для аналитиков, которым нужно иметь доступ к просмотру данных в базе данных. Создание пользователя на рисунке 2.5.

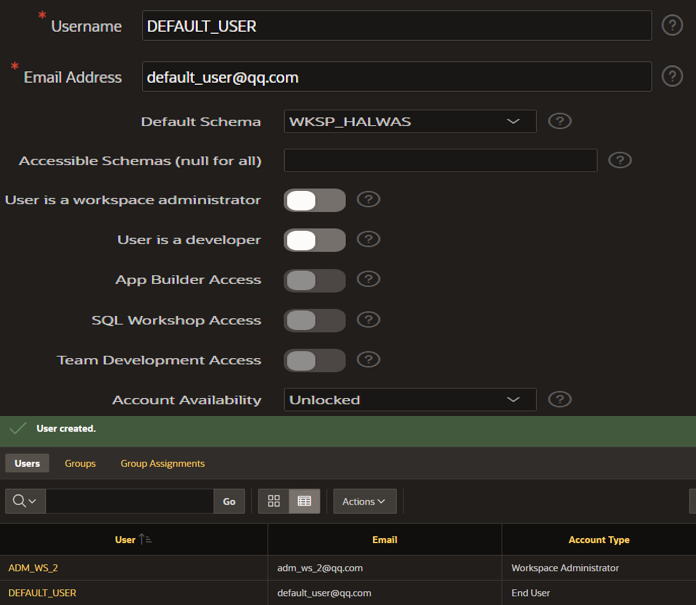


Рисунок 2.5 – Создание пользователя сервиса

Как видно, пользователь default\_user не имеет никаких прав на участие в разработке проекта.

Дальнейшие действия по созданию инфраструктуры базы данных будут выполняться пользователем, имеющим лишь права разработчика базы данных.

## **Разработка таблиц базы данных**

Для текущей базы данных были разработаны таблицы, связанные между собой полями, имеющими первичный и внешний ключи.

Разберём составленные таблицы.

Таблица 2.1 «Departs»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Значение поля |
| D\_ID | VARCHAR2(12) | Первичный ключ |
| D\_NAME | VARCHAR2(100) | Название отдела |

В компании, занимающейся доставкой товаров, работают разные сотрудники с разными должностями. Для этого, следует разработать таблицу, которая отвечает за отделы, в которых состоят сотрудники.

Таблица 2.2 «Posts»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Значение поля |
| P\_POSTS | VARCHAR2(30) | Первичный ключ |
| P\_SAL | NUMBER(8,2) | Зарплата на текущей должности |

Таблица отвечает за должности сотрудников. Причём, при создании таблицы для данного поля было указано, что в нём могут содержаться лишь значения, удовлетворяющие условию P\_SAL > 450.

Таблица 2.3 «Employees»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Значение поля |
| E\_ID | NCHAR(4) | Первичный ключ |
| E\_FNAME | VARCHAR2(25) | Имя сотрудника |
| E\_BORN | DATE | Дата рождения |
| E\_SEX | CHAR(1) | Пол сотрудника |
| E\_PASP | CHAR(10) | Серия и номер паспорта сотрудника |
| E\_DATE | DATE | Дата получения паспорта |
| E\_GIVEN | VARCHAR2(50) | Кем выдан паспорт |
| E\_DEPART | VARCHAR2(12) | Отдел, в котором состоит сотрудник; внешний ключ к DEPARTS (D\_ID) |
| E\_POST | VARCHAR2(30) | Должность сотрудника; внешний ключ к POSTS (P\_POSTS) |

Таблица «Employees» отвечает за хранение подробной информации о сотрудниках компании. Некоторые поля этой таблицы служат для связи с другими таблицами.

Таблица 2.4 «ADRTEL»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Значение поля |
| A\_ID | NCHAR(4) | Внешний ключ к Employees E\_ID) |
| A\_ADDR | VARCHAR2(50) | Адрес |
| A\_PHONE | VARCHAR2(30) | Номер телефона |

Таблица, необходимая для приведения спроектированной базы данных к нормальной форме. Содержит в себе дополнительную информацию о сотрудниках.

Таблица 2.5 «CLIENTS»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Значение поля |
| C\_ID | NCHAR(4) | Первичный ключ |
| C\_CUSTOMER | VARCHAR2(40) | Заказчик |
| C\_ADR | VARCHAR2(50) | Адрес заказчика |
| C\_PERSON | VARCHAR2(50) | Контактное лицо |
| C\_PHONE | VARCHAR2(30) | Телефон |

Таблица необходима для хранения данных о клиентах.

Таблица 2.6 «ORDERS»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Значение поля |
| O\_ABBR | CHAR(10) | Первичный ключ |
| O\_TITLE | VARCHAR2(100) | Название проекта |
| O\_DEPART | VARCHAR2(12) | Отдел; внешний ключ к DEPARTS (D\_ID) |
| O\_CUSTOMER | NCHAR(4) | Заказчик; внешний ключ к CLIENTS (C\_ID) |
| O\_BEGIN | DATE | Дата начала заказа |
| O\_END | DATE | Дата окончания заказа |
| O\_COST | NCHAR(10) | Стоимость заказа |

Таблица содержит в себе информацию о заказах.

Таблица 2.7 «STAGES»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Значение поля |
| S\_ORD | CHAR(10) | Внешний ключ к ORDERS (O\_ABBR) |
| S\_NUM | NUMBER(2,0) | Номер этапа |
| S\_TITLE | VARCHAR2(100) | Название этапа |
| S\_BEGIN | DATE | Дата начала этапа |
| S\_END | DATE | Дата окончания этапа |
| S\_COST | NCHAR(10) | Стоимость этапа |
| S\_SUM | NCHAR(10) | Конечная сумма |

Таблица, предназначенная для хранения информации об этапах выполнения заказа.

Таблица 2.8 «JOB»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Значение поля |
| J\_PRO | CHAR(10) | Внешний ключ к ORDERS (O\_ABBR) |
| J\_EMP | NCHAR(4) | Внешний ключ к EMPLOYEES (E\_ID) |
| J\_ROLE | VARCHAR2(20) | Роль сотрудника |
| J\_BONUS | NUMBER(2,0) | Бонус |

Таблица, предназначенная для хранения дополнительной информации о заказах и сотрудниках, которые выполняют соответствующие заказы.

## **Диаграммы UML**

Также, будет не лишним разработать диаграмму сущностей, на которой удобно расположить все сущности, их содержимое и связи с другими сущностями. Для разработки таких диаграмм могут быть использованы разные сервисы. Данная диаграмма была построена при помощи сервиса draw.io. Диаграмма, помимо сущностей и наличия связей, также должна отображать и типы связей. Бывают разные связи: один к одному, один ко многим и многие ко многим. Диаграмма сервиса демонстрируется а рисунке 2.6.

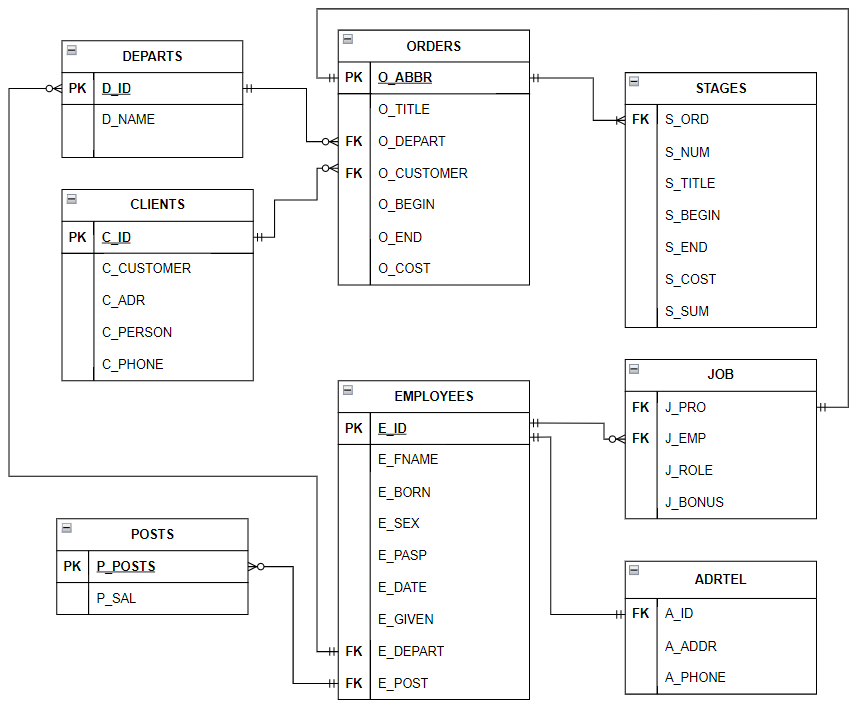


Рисунок 2.6 – Диаграмма сущностей

На диаграмме сущностей отображены все типы зависимостей таблиц.

Также, следует представить диаграмму вариантов использования продукта, на которой должны быть чётко отображены те возможности, которые доступны тому или иному пользователю с его ролью, будь это обычный пользователь или администратор. С данной диаграммой намного проще воспринимаются возможности ролей, так как каждая роль может так или иначе расширять возможности другой. Диаграмма вариантов использования представлена на рисунке 2.7.

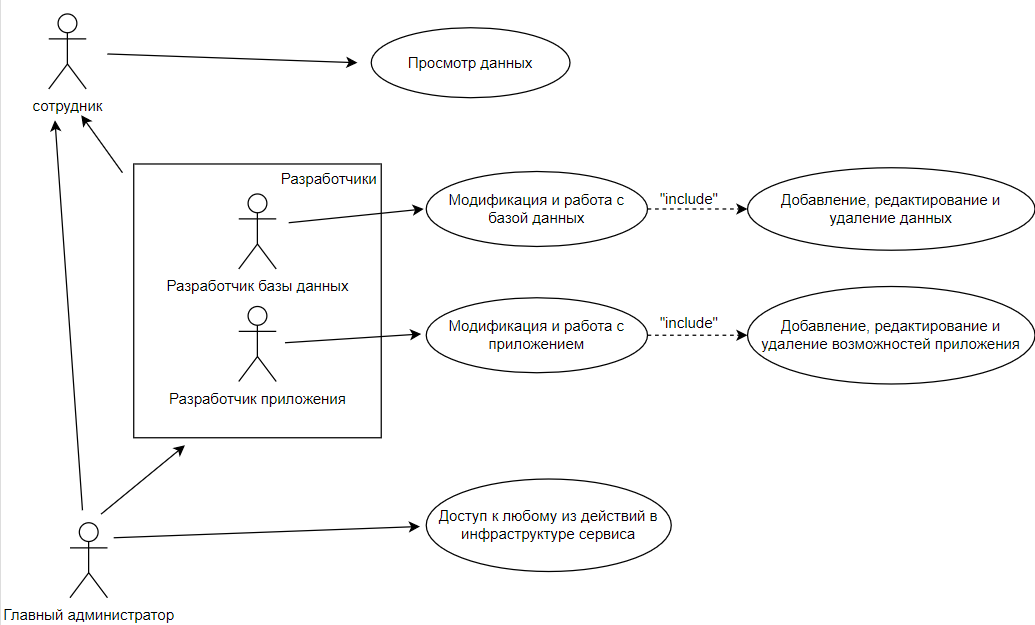


Рисунок 2.7 – Диаграмма вариантов использования

Данная диаграмма наглядно демонстрирует возможности ролей, существующих на данный момент.

# **Разработка объектов базы данных**

## **Последовательности**

Для начала, стоит задуматься о том, чтобы значения числовых идентификаторов в таблицах автоматически инкрементировались. Для этого, на помощь приходят последовательности.

Создадим последовательность для наших таблиц.

|  |
| --- |
| create sequence int\_sequence  start with 1  increment by 1  NOCACHE  NOCYCLE; |

Листинг 3.1 – Последовательность для таблиц

Указываем, что наша последовательность начинается с единицы и инкрементируется на единицу. Такая последовательность может быть использована при вставке значений в таблицы с первичным ключом целочисленного типа. Также указываем NOCACHE и NOCYCLE, чтобы гарантировать непрерывную нумерацию.

В oracle создавать объекты такого типа очень удобно, т.к. все недостающие данные он допишет автоматически, что показано на рисунке 3.1.

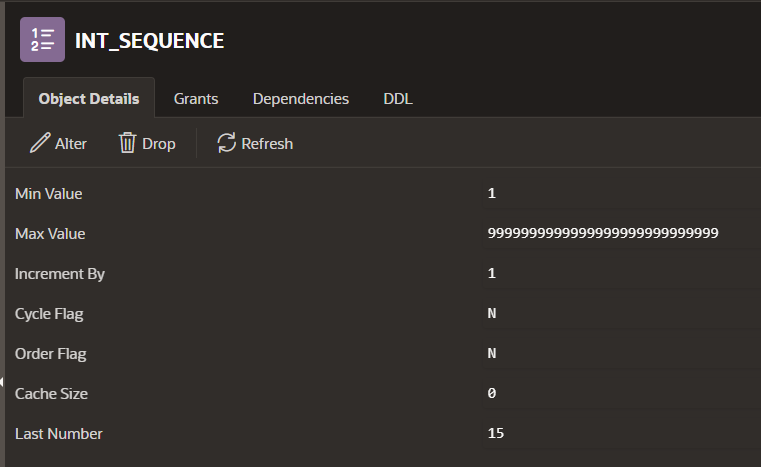


Рисунок 3.1 - Созданная последовательность

Как видно, при необходимости, можно динамически менять значения в создаваемых последовательностях.

## **Представления**

Представление является хранимой инструкцией SELECT. Проще говоря, представление – это поименованный select-запрос. Представления запрашиваются как таблицы и не принимают параметры. Один из способов создания на рисунке 3.2.

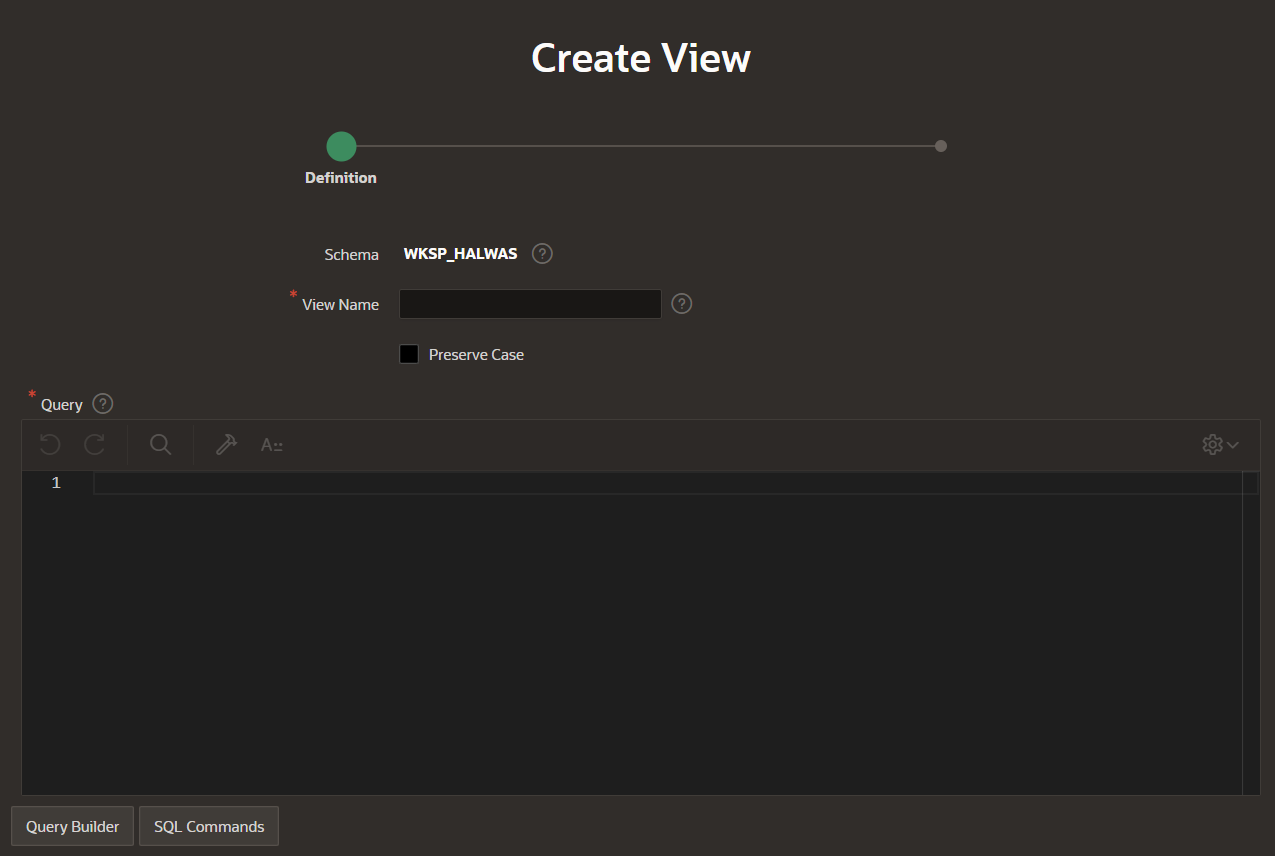


Рисунок 3.2 – Встроенный способ создания представлений в Oracle Apex

Существуют некоторые ограничения при работе с представлениями:

* нельзя повесить триггер на представление,
* нельзя сделать представление на основе временных таблиц; нельзя сделать временное представление,
* в определении представления нельзя использовать подзапрос в части FROM,
* в определении представления нельзя использовать системные и пользовательские переменные; внутри хранимых процедур нельзя в определении представления использовать локальные переменные или параметры процедуры,
* в определении представления нельзя использовать параметры подготовленных выражений (PREPARE),
* таблицы и представления, присутствующие в определении представления, должны существовать,
* только представления, удовлетворяющие ряду требований, допускают запросы типа UPDATE, DELETE и INSERT.

Для начала, неплохо было бы создать какое-нибудь представление для работы с таблицей, содержащей информацию о сотрудниках.

|  |
| --- |
| CREATE VIEW ADRTEL\_BY\_ADDR as select A\_ID, A\_ADDR from ADRTEL order by A\_ADDR |

Листинг 3.2 – Представление ADRTEL\_BY\_ADDR

Представление ADRTEL\_BY\_ADDR будет ссылаться на таблицу с адресами и телефонами сотрудников, а следующее будет выводить сотрудников в отсортированном по дате рождения виде.

|  |
| --- |
| CREATE VIEW EMPLOYEES\_BY\_BORN as se1ect Е\_FNAME from EMPLOYEES order by E\_BORN |

Листинг 3.3 – Представление EMPLOYEES\_BY\_BORN

Представление EMPLOYEES\_BY\_BORN поможет ускорить поиск, в котором буде использовать таблица сотрудников.

Выведем результат работы представления на рисунке 3.3.

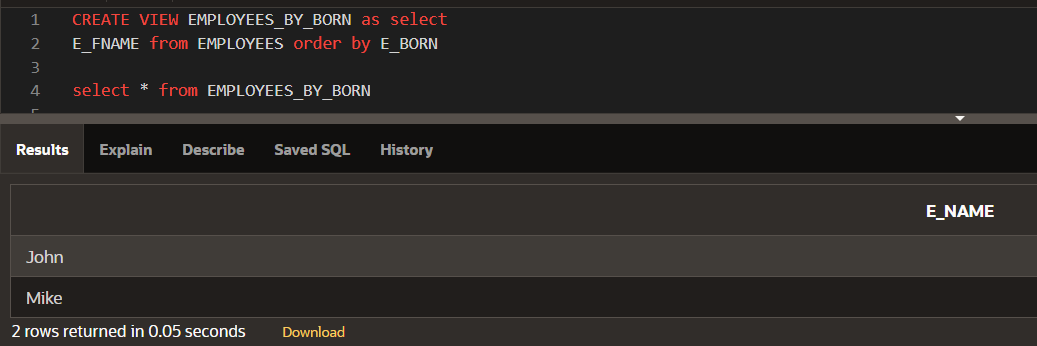


Рисунок 3.3 – Работа представления о сотрудниках

Выведем, для наглядности, даты рождения сотрудников.

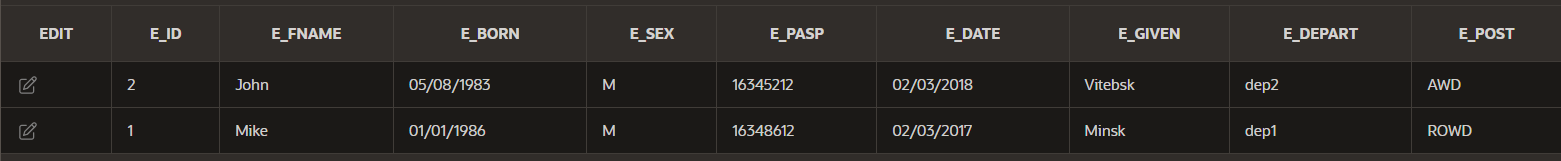


Рисунок 3.4 – Демонстрация правильной работы сортировки в представлении

Как видим, сортировка в представлении работает должным образом – получен результат, в котором выводятся, сначала, сотрудники возрастом старше.

Также, взглянем на то, как работает представление ADRTEL\_BY\_ADDR

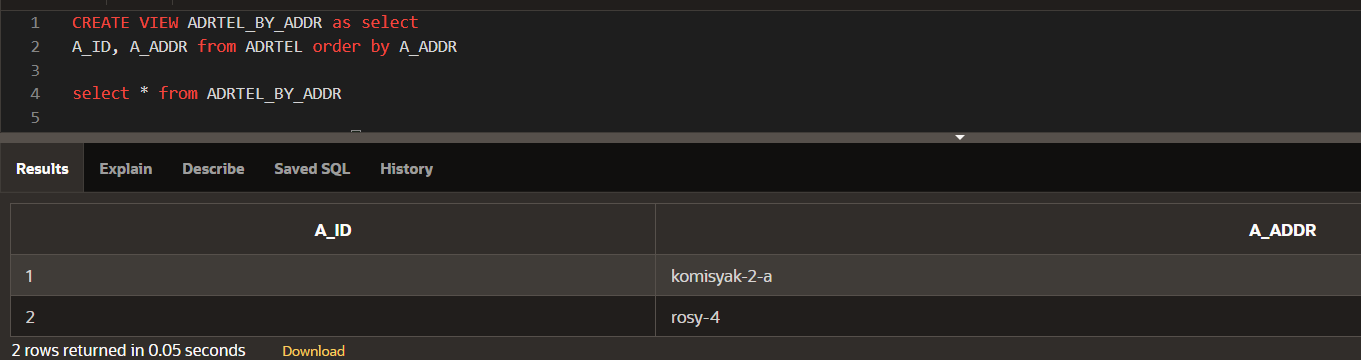


Рисунок 3.5 – Работа представления об адресах проживания сотрудников

С этим представлением также нет никаких проблем.

## **Триггеры**

Триггер — это особая разновидность хранимой процедуры, которая автоматически выполняется при возникновении события на сервере базы данных. Триггеры DML выполняются, когда пользователь пытается изменить данные с помощью событий языка обработки данных (DML).

Создадим триггер для обновления адреса сотрудника при обновлении его отдела:

|  |
| --- |
| CREATE TRIGGER tr\_emp\_depart\_update  AFTER UPDATE OF E\_DEPART ON employees  FOR EACH ROW  BEGIN  IF :NEW.E\_DEPART != :OLD.E\_DEPART THEN  UPDATE adrtel SET A\_ADDR = 'New Department: ' || :NEW.E\_DEPART, A\_PHONE = A\_PHONE WHERE A\_ID = :NEW.E\_ID;  END IF;  END; |

Листинг 3.4 – Триггер для обновления таблицы employees

Этот триггер будет обновлять адрес сотрудника, включая его новый отдел, каждый раз, когда его отдел обновляется.

Будет полезным создать триггер для предотвращения удаления отдела с сотрудниками:

|  |
| --- |
| CREATE TRIGGER tr\_dep\_delete  BEFORE DELETE ON departs  FOR EACH ROW  DECLARE  emp\_count int;  BEGIN  SELECT COUNT(\*) INTO emp\_count FROM employees  WHERE E\_DEPART =: OLD.d\_id;  IF emp\_count > 0 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Cannot delete department with employees');  END IF;  END; |

Листинг 3.5 – Триггер для предотвращения удаления таблицы отдела с сотрудниками

Этот триггер срабатывает перед выполнением операции удаления из таблицы departs. Сначала он проверяет, есть ли сотрудники, связанные с удаляемым отделом. Если таковые имеются, он выдает ошибку приложения с сообщением "Невозможно удалить отдел с сотрудниками" и откатывает транзакцию.

Процедура RAISE\_APPLICATION\_ERROR используется для создания ошибки с заданным пользователем кодом ошибки и сообщением. В данном случае код ошибки -20001.

Также, создадим триггер для обновления стоимости заказа при каждом обновлении его стадий. Стоит отметить, что oracle предоставляет возможность отслеживать изменения данных и разграничивать их на так называемые старые и новые. За счёт конструкций OLD и NEW можно реализовать данный функционал.

|  |
| --- |
| CREATE TRIGGER tr\_stage\_update  AFTER INSERT OR UPDATE OR DELETE ON stages  FOR EACH ROW  DECLARE  v\_order\_cost NUMBER;  BEGIN  SELECT SUM(s\_cost) INTO v\_order\_cost FROM stages  WHERE s\_ord =: NEW.s\_ord;  UPDATE orders SET o\_cost = v\_order\_cost WHERE o\_abbr =: NEW.s\_ord;  END; |

Листинг 3.6 – Создание триггрера для расчёта новой стоимости заказа после изменений в таблице стадий

Этот триггер срабатывает после выполнения операции вставки, обновления или удаления в таблице этапов. Затем он вычисляет сумму столбца s\_cost для всех этапов, связанных с обновляемым заказом. Наконец, он обновляет столбец o\_cost в таблице заказов рассчитанной суммой. Функция SUM используется для вычисления суммы столбца s\_cost для всех этапов, связанных с обновляемым заказом.

Протестируем какой-нибудь из триггеров. Например, триггер, предотвращающий удаление отделов:

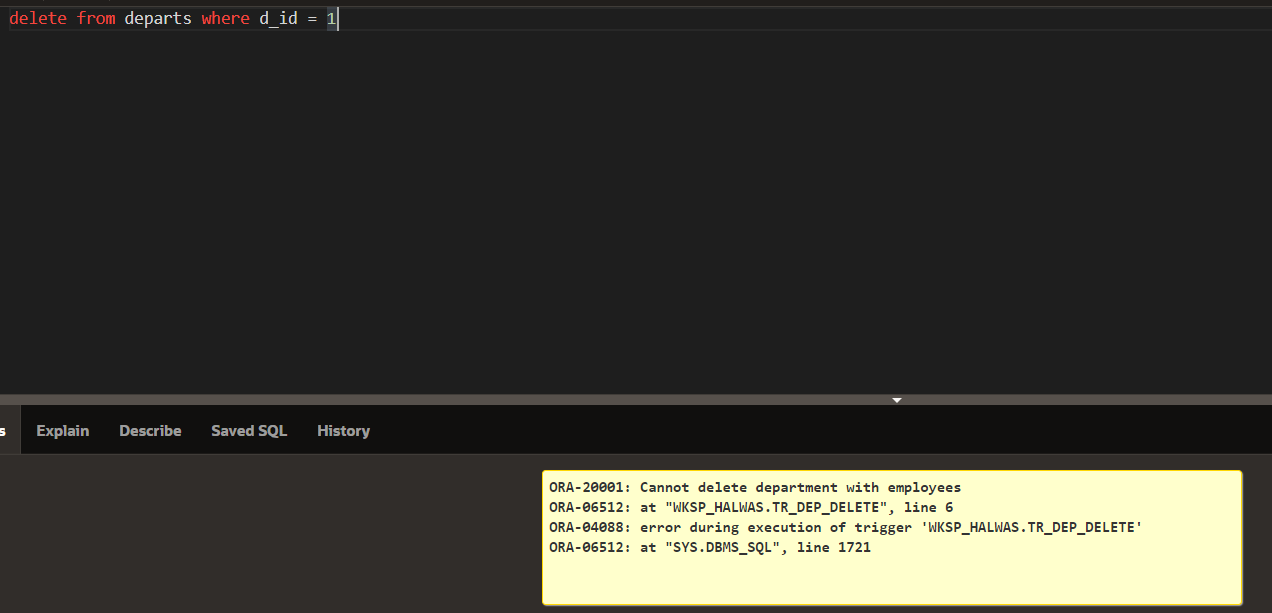


Рисунок 3.6 – Тестирование одного из триггеров

Как можно заметить, триггер сработал как надо и выдал исключение.

## **Индексы**

Индексы — это структура данных, которая помогает сократить время поиска запрошенных данных. Индексы достигают этого за счет дополнительных затрат на хранение, память и поддержание их в актуальном состоянии (более медленная запись), что позволяет пропустить утомительную проверку каждой строки таблицы.

Важно отметить, что эффективность индекса может варьироваться в зависимости от конкретного запроса, размера таблицы и распределения данных.

Создадим индекс для столбцов E\_DEPART и E\_POST: Этот индекс поможет улучшить производительность запросов, которые соединяют таблицы employees и departs по столбцам E\_DEPART и d\_id соответственно, а также для запросов между таблицами emloyees и posts:

|  |
| --- |
| CREATE INDEX idx\_employees\_dept\_post ON employees (E\_DEPART, E\_POST); |

Листинг 3.7 – Индекс idx\_empoloyees\_depart

Будет полезен индекс на orders.o\_customer. Этот индекс поможет улучшить производительность запросов, которые соединяют таблицы orders и clients по столбцам o\_customer и c\_id соответственно:

|  |
| --- |
| CREATE INDEX idx\_orders\_customer ON orders (o\_customer); |

Листинг 3.8 – Индекс idx\_orders\_customer

Стадии и заказы – наиболее часто используемые сущности нашей базы данных. Создадим индекс на stages.s\_ord. Этот индекс поможет улучшить производительность запросов, которые соединяют таблицы stages и orders по столбцам s\_ord и o\_abbr соответственно:

|  |
| --- |
| CREATE INDEX idx\_stages\_order ON stages (s\_ord); |

Листинг 3.9 – Индекс idx\_stages\_order

Ну, и, наконец, создадим индекс на job.J\_EMP. Этот индекс поможет улучшить производительность запросов, которые соединяют таблицы job и employees по столбцам J\_EMP и E\_ID соответственно:

|  |
| --- |
| CREATE INDEX idx\_job\_employee ON job (J\_EMP); |

Листинг 3.10 – Индекс для таблиц job и employees

Данный индекс ускорит поиск, задействующий данную таблицу.

## **Материализованное представление**

Материализованное представление – это физическое представление данных, которое хранит результаты запроса и обновляется автоматически при изменении базовых данных. Это может улучшить производительность запросов, особенно при частых запросах на агрегированные данные. Поэтому, перед созданием таких объектов базы данных следует анализировать и оценивать его потенциальную пользу и затраты на ресурсы.

Важно отметить, что материализованные представления занимают дополнительное место на диске и могут замедлять время выполнения DML-операций на базовых таблицах.

Для создания материализованного представления разумно будет выполнить следующие шаги:

1. Создадим таблицу для хранения результата запроса:

|  |
| --- |
| CREATE TABLE my\_materialized\_view (  o\_customer nchar(4),  total\_cost nchar(10)  ); |

Листинг 3.11 – Создание таблицы для материализованного представления

1. Создание материализованного представления:

|  |
| --- |
| CREATE MATERIALIZED VIEW LOG ON ORDERS  WITH ROWID, SEQUENCE (o\_abbr, o\_depart, o\_customer, o\_begin, o\_end, o\_cost)  INCLUDING NEW VALUES; |

Листинг 3.12 – Создание материализованного журнала для таблицы заказов

Для создания материализованного представления необходимо сначала создать материализованный журнал для таблицы "ORDERS". Материализованный журнал позволит Oracle отслеживать изменения в таблице и обновлять материализованное представление в соответствии с этими изменениями

Этот запрос создаст материализованный журнал для таблицы "ORDERS" и будет отслеживать изменения в указанных столбцах. Ключевое слово "INCLUDING NEW VALUES" позволяет включить в журнал новые значения, которые будут вставлены в таблицу.

1. Создание материализованных представлений.

Создадим материализованное представление materialized\_view\_sum\_cost, которое содержит суммарную стоимость заказов для каждого клиента.

|  |
| --- |
| CREATE MATERIALIZED VIEW materialized\_view\_sum\_cost  BUILD IMMEDIATE  REFRESH FAST ON COMMIT  AS  SELECT o\_customer, SUM(o\_cost) as total\_cost  FROM orders  GROUP BY o\_customer; |

Листинг 3.13 – Создание материазизованного представления

Такие данные как раз часто используются сотрудниками компаний для анализа покупательской способности клиентов.

Воспользуемся созданным материализованным представлением:

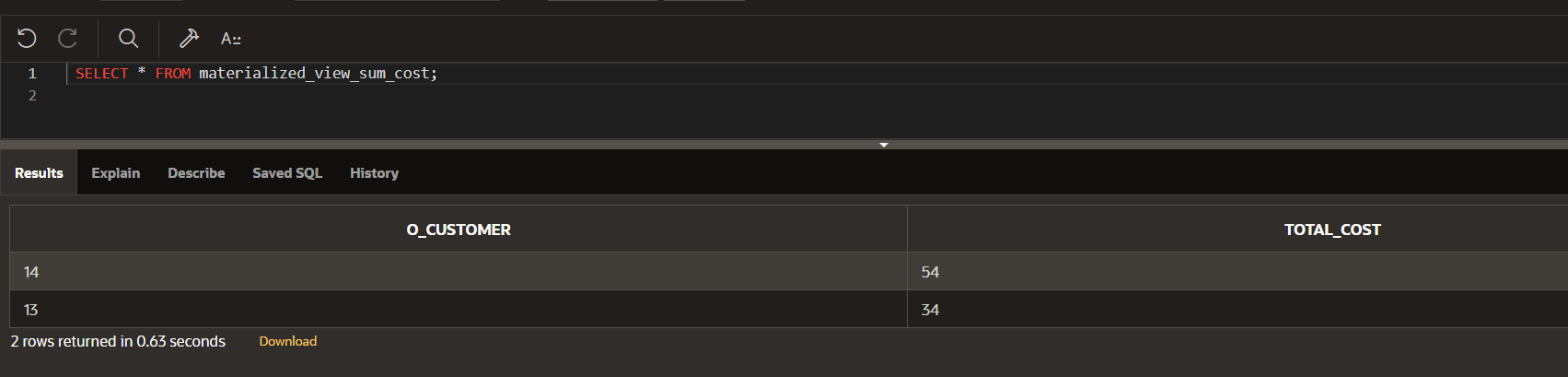


Рисунок 3.7 – Демонстрация работы материализованного представления

Как можно заметить, в соответствующие поля выводятся необходимые данные.

## **Хранимые процедуры**

Хранимая процедура – это скомпилированный набор SQL-предложений, сохраненный в базе данных как именованный объект и выполняющийся как единый фрагмент кода. Хранимые процедуры могут принимать и возвращать параметры. Когда пользователь создает хранимую процедуру, сервер компилирует ее и помещает в разделяемый кэш, после чего скомпилированный код может быть применен несколькими пользователями. Когда приложение использует хранимую процедуру, оно передает ей параметры, если таковые требуются, и сервер выполняет процедуру без перекомпиляции.

Хранимые процедуры позволяют повысить производительность приложений. Во-первых, по сравнению с обычными SQL-запросами, посылаемыми из клиентского приложения, они требуют меньше времени для подготовки к выполнению, поскольку они уже скомпилированы и сохранены. Во-вторых, сетевой трафик в этом случае также меньше, чем в случае передачи SQL-запроса, так как по сети передается меньшее количество данных.

Создадим процедуры, которые могут быть использованы чаще всего:

1. Процедура для добавления сотрудников:

|  |
| --- |
| CREATE OR REPLACE PROCEDURE add\_employee(  i\_id IN employees.E\_ID%TYPE, i\_fname IN employees.E\_fname%TYPE,  i\_born IN employees.E\_BORN%TYPE, i\_sex IN employees.E\_SEX%TYPE,  i\_pasp IN employees.E\_PASP%TYPE, i\_date IN employees.E\_DATE%TYPE,  i\_given IN employees.E\_GIVEN%TYPE, i\_depart IN employees.E\_DEPART%TYPE, i\_post IN employees.E\_POST%TYPE  ) AS BEGIN INSERT INTO employees(E\_ID, E\_fname, E\_BORN, E\_SEX, E\_PASP, E\_DATE, E\_GIVEN, E\_DEPART, E\_POST) VALUES (i\_id, i\_fname, i\_born, i\_sex, i\_pasp, i\_date, i\_given, i\_depart, i\_post); COMMIT; END; |

Листинг 3.14 – Процедура добавления сотрудников

Операции рода вставка, удаление и обновление рекомендуется проводить, используя оператор commit, чтобы можно было фиксировать изменения в базе данных и, при необходимости, откатываться к предыдущему состоянию.

Протестируем выполнение процедуры add\_employee для таблицы сотрудников:

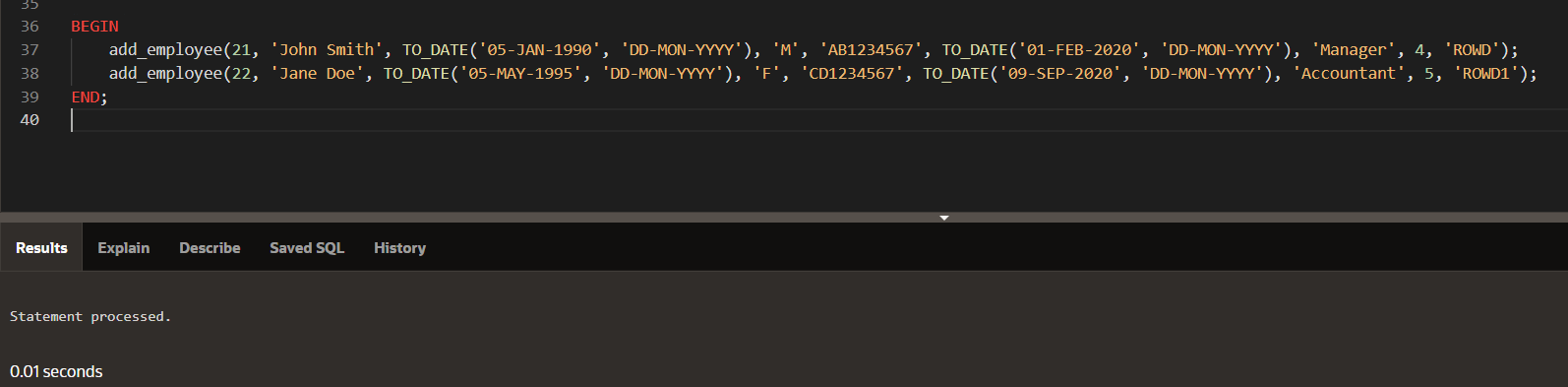


Рисунок 3.8 – Работа процедуры add\_employee

Как видно, данные вставляются без каких-либо проблем.

Но, на этом останавливаться не стоит. Стоит добавить ещё несколько процедур, которые упростят работу базой данных.

1. Процедура обновления информации о сотруднике:

|  |
| --- |
| CREATE OR REPLACE PROCEDURE update\_employee (  p\_id IN INTEGER,  p\_name IN VARCHAR2,  p\_department IN INTEGER,  p\_post IN VARCHAR2  ) AS  BEGIN  UPDATE employees  SET e\_fname = p\_name,  e\_depart = p\_department,  e\_post = p\_post  WHERE e\_id = p\_id;  END; |

Листинг 3.15 – Обновление информации о сотрудниках

1. Процедура выбора сотрудников по определённому фильтру:

|  |
| --- |
| CREATE OR REPLACE PROCEDURE get\_records\_by\_department (  p\_department IN INTEGER,  p\_cursor OUT SYS\_REFCURSOR  )  AS BEGIN  OPEN p\_cursor FOR  SELECT \* FROM employees  WHERE e\_depart = p\_department;  END; |

Листинг 3.16 – Выбор сотрудника из таблицы

Эта процедура ищет записи по переданному в неё значению, являющейся идентификатором.

1. Процедуры для взаимодействия с таблицей adrtel:

|  |
| --- |
| CREATE OR REPLACE PROCEDURE insert\_adrtel(  p\_A\_ID IN adrtel.A\_ID%TYPE,  p\_A\_ADDR IN adrtel.A\_ADDR%TYPE,  p\_A\_PHONE IN adrtel.A\_PHONE%TYPE  )  AS  BEGIN  INSERT INTO adrtel(A\_ID, A\_ADDR, A\_PHONE)  VALUES (p\_A\_ID, p\_A\_ADDR, p\_A\_PHONE);  COMMIT;  END;  /  CREATE OR REPLACE PROCEDURE update\_adrtel\_phone(  p\_A\_ID IN adrtel.A\_ID%TYPE,  p\_A\_PHONE IN adrtel.A\_PHONE%TYPE  )  AS  BEGIN  UPDATE adrtel  SET A\_PHONE = p\_A\_PHONE  WHERE A\_ID = p\_A\_ID;  COMMIT;  END;  /  CREATE OR REPLACE PROCEDURE delete\_adrtel(  p\_A\_ID IN adrtel.A\_ID%TYPE  )  AS  BEGIN  DELETE FROM adrtel  WHERE A\_ID = p\_A\_ID;  COMMIT;  END;  / |

Листинг 3.17 – Процедуры для работы с таблицей adrtel

Скрипт в данном листинге выполняет элементарные операции вставки, удаления и обновления информации.

Это не все процедуры для работы с базой данных, а лишь основные. Больше процедур находится в приложении.

## **Функции**

Функции SQL в Oracle можно разделить на несколько категорий:

* Однострочные функции (single-row functions) - применяются для выполнения задач вроде преобразования слова из нижнего регистра в верхний или наоборот или замены какой-нибудь части текста в строке.
* Агрегатные функции SQL - используются для выполнения операций над группами строк в таблице.
* Числовые и календарные функции SQL - используются для выполнения операций с числами и датами.
* Универсальные и условные функции SQL - используются для выполнения различных операций с данными в базе данных.
* Аналитические функции SQL - используются для выполнения аналитических операций над данными в базе данных.

Создадим некоторые функции, которые, в дальнейшем, упростят работу с нашей базой данных.

Первая функция, которую мы создадим, поможет нам считать количество сотрудников в каждом отделе:

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_employee\_count\_by\_dept  RETURN SYS\_REFCURSOR IS my\_cursor SYS\_REFCURSOR;  BEGIN  OPEN my\_cursor FOR  SELECT d.d\_name, COUNT(\*) AS employee\_count  FROM departs d  JOIN employees e ON d.d\_id = e.e\_depart  GROUP BY d.d\_name;  RETURN my\_cursor;  END;  / |

Листинг 3.18 – Функция подсчёта сотрудников, относящихся к определённому отделу

Эта функция возвращает курсор, который содержит названия отделов и количество сотрудников в каждом отделе.

Для тестирования нашей функции воспользуемся курсором, который необходимо, для начала, написать.

|  |
| --- |
| DECLARE  my\_cursor SYS\_REFCURSOR;  dept\_name VARCHAR2(100);  emp\_count NUMBER;  BEGIN  my\_cursor := get\_employee\_count\_by\_dept;  LOOP  FETCH my\_cursor INTO dept\_name, emp\_count;  EXIT WHEN my\_cursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(dept\_name || ': ' || emp\_count || ' employees');  END LOOP;  CLOSE my\_cursor;  END;  / |

Листинг 3.19 – Курсор для тестирования функции get\_employee\_count\_by\_dept

Выведем результат работы курсора:

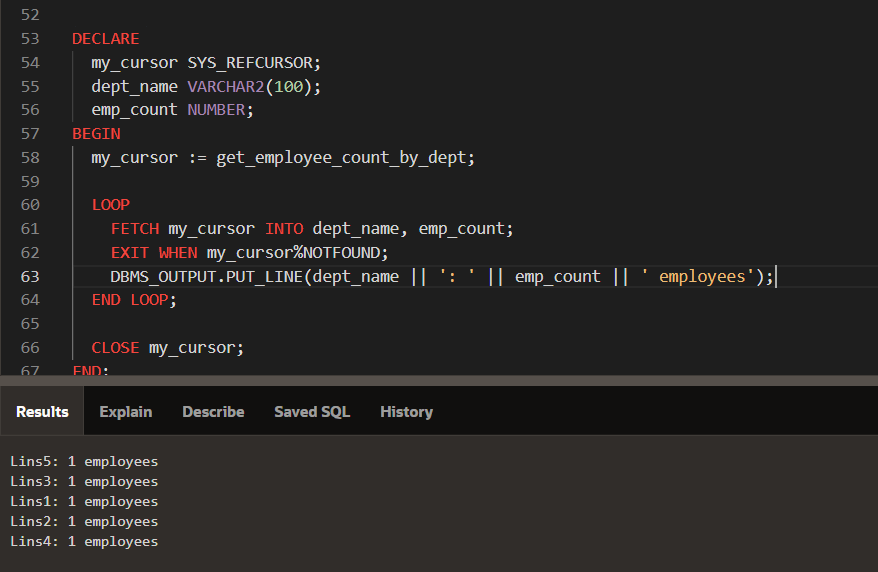


Рисунок 3.9 – Демонстрация работы функции

Каждый пользователь сервиса в тот или иной момент совершает заказывает какой-либо товар, услугу и прочее. Эти действия происходят в определённый момент времени. Следовательно, крайне необходимо иметь возможность отслеживания совершения заказов пользователями в определённый день. Создадим функцию, предоставляющую такой функционал:

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_customers\_by\_date(p\_order\_date IN DATE) RETURN SYS\_REFCURSOR IS  my\_cursor SYS\_REFCURSOR;  BEGIN  OPEN my\_cursor FOR  SELECT c.c\_customer, c.c\_adr, c.c\_person, c.c\_phone  FROM clients c  JOIN orders o ON  c.c\_id = o.o\_customer  WHERE o.o\_begin = p\_order\_date;  RETURN my\_cursor;  END;  / |

Листинг 3.20 – Функция для получения списка клиентов, совершивших заказ за определённую дату

Эта функция возвращает курсор, который содержит список клиентов, которые сделали заказы на заданную дату.

Реализуем этот курсор:

|  |
| --- |
| DECLARE  v\_customer clients.c\_customer%TYPE;  v\_adr clients.c\_adr%TYPE;  v\_person clients.c\_person%TYPE;  v\_phone clients.c\_phone%TYPE;  my\_cursor SYS\_REFCURSOR;  BEGIN  my\_cursor := get\_customers\_by\_date(TO\_DATE('2020-08-09', 'YYYY-MM-DD'));  LOOP  FETCH my\_cursor INTO v\_customer, v\_adr, v\_person, v\_phone;  EXIT WHEN my\_cursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(v\_customer || ' ' || v\_adr || ' ' || v\_person || ' ' || v\_phone);  END LOOP;  CLOSE my\_cursor;  END;  / |

Листинг 3.21 – Реализация курсора для использования функции get\_customers\_by\_date

Продемонстрируем работу этих функции и курсора.

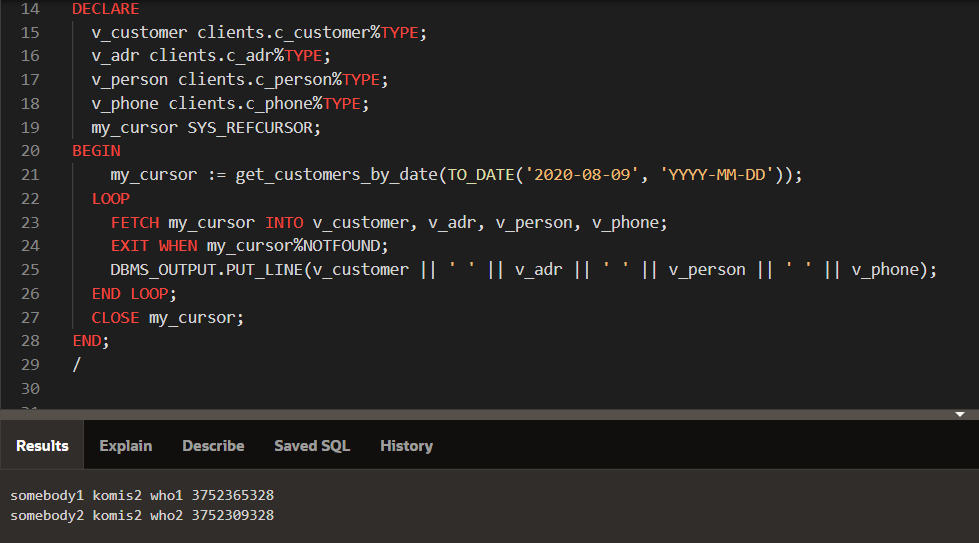


Рисунок 3.10 – Курсор для функции get\_customers\_by\_date

Предположим, нам нужно получить подробную информацию о заказе, то есть, узнать, кем был совершён заказ, время выполнения заказа, имя клиента, адрес доставки и конечная сумма заказа.

Данная функция будет использовать внутреннее соединение необходимых ей таблиц для получения информации об определённом заказе, уникальный идентификатор которого должен быть указан при вызове данной функции. Для получения продолжительности заказа используется разница значений дат, которую можно получить за счёт функции Trunc(). Напрямую значения типа данных DATE вычитать нельзя, но, благодаря функции Trunc() можно привести значение даты к типу данных number. Соответственно, выполнив такие действия, можно будет получить разницу дат в часах.

Также, стоит учесть, что условия данной функции, в будущем, могут измениться. Для этого существует очень удобная конструкция, которая позволяет, при необходимости перезаписать наш объект базы данных. Она называется replace. При необходимости можно использовать конструкцию create or replace, что и будет сделано.

Реализуем запрос с такими требованиями:

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_order\_info (employee\_id IN employees.e\_id%TYPE)  RETURN SYS\_REFCURSOR  IS  c1 SYS\_REFCURSOR;  BEGIN  OPEN c1 FOR  SELECT o.o\_title, d.d\_name, c.c\_customer, c.c\_adr,  TRUNC(o.o\_end) - TRUNC(o.o\_begin) AS duration,  s.s\_sum AS total\_cost  FROM orders o  JOIN departs d ON o.o\_depart = d.d\_id  JOIN clients c ON o.o\_customer = c.c\_id  JOIN stages s ON o.o\_abbr = s.s\_ord  JOIN job j ON o.o\_abbr = j.j\_pro  WHERE j.j\_emp = employee\_id  GROUP BY o.o\_title, d.d\_name, c.c\_customer, c.c\_adr,  TRUNC(o.o\_end) - TRUNC(o.o\_begin), s.s\_sum;  RETURN c1;  END;  / |

Листинг 3.22 – Создание функции получения подробной информации о заказе

Для вызова данной функции будем использовать следующий курсор:

|  |
| --- |
| DECLARE  c1 SYS\_REFCURSOR;  title orders.o\_title%TYPE;  depart departs.d\_name%TYPE;  customer clients.c\_customer%TYPE;  adr clients.c\_adr%TYPE;  duration NUMBER;  total\_cost nchar(10);  BEGIN  c1 := get\_order\_info('1');  LOOP  FETCH c1 INTO title, depart, customer, adr, duration, total\_cost;  EXIT WHEN c1%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE('Order: ' || title || ', Depart: ' || depart || ', Customer: ' || customer || ', Address: ' || adr || ', Duration: ' || duration || ' hours' || ', Total Cost: ' || total\_cost);  END LOOP;  CLOSE c1;  END;  / |

Листинг 3.23 – Курсор для вызова функции

Продемонстрируем работу данных объектов:

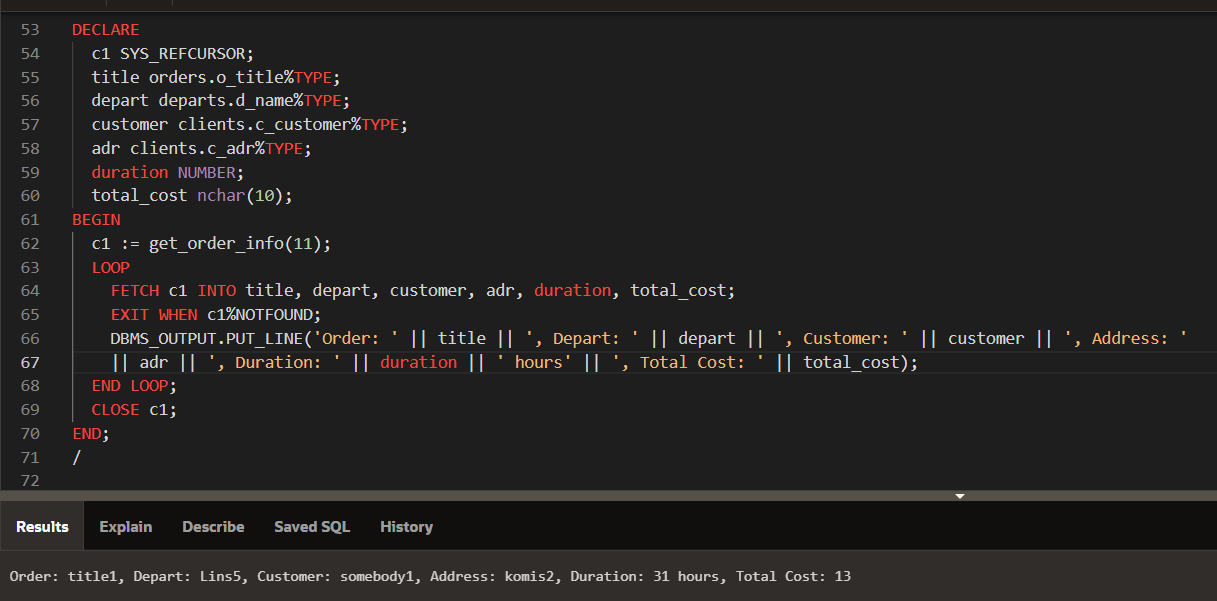


Рисунок 3.11 – Реализация функции get\_order\_info в курсоре

Как можно заметить, вывод результата выполнения работает без проблем.

## **Вывод по разделу**

Разработка таких объектов базы данных – важнейшая часть разработки сервиса. Как минимум, за счёт таких объектов работа с базой данных будет упрощена в разы, а время, затрачиваемое на написание sql-запросов, может быть сведено к минимуму.

# **Описание процедур импорта и экспорта**

Процедуры экспорта и импорта подразумевают перенос данных из одной базы данных в другую или для резервного копирования. Единственный способ быть уверенным в восстановлении данных — самостоятельно выполнять регулярное резервное копирование информации.

Также, можно сказать, что данные процедуры выгружают или загружают данные из либо в базу данных с преобразованием формата.

Данные задачи можно реализовывать по-разному. Благодаря используемой в рамках данного курсового проекта технологии, это можно сделать с помощью встроенных в Oracle Apex средств:

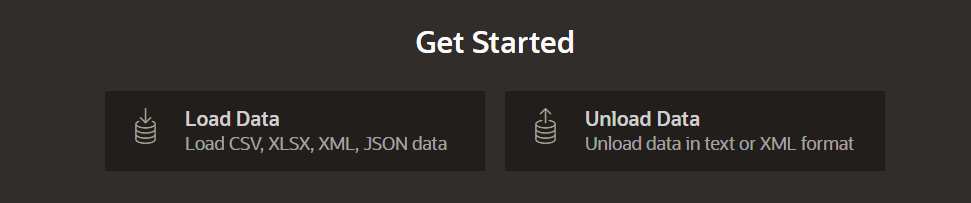


Рисунок 4.1 – Встроенные технологии Apex для импорта и экспорта данных

Как можно заметить, экспорт доступен лишь в XML-формате, однако импортировать можно из большего числа форматов.

## **Импорт данных**

Важно учитывать, что данная технология позволяет выгружать лишь таблицы с имеющимися в них данными. Никакие другие объекты базы данных экспортировать или импортировать не получиться, за исключением материализованных представлений.

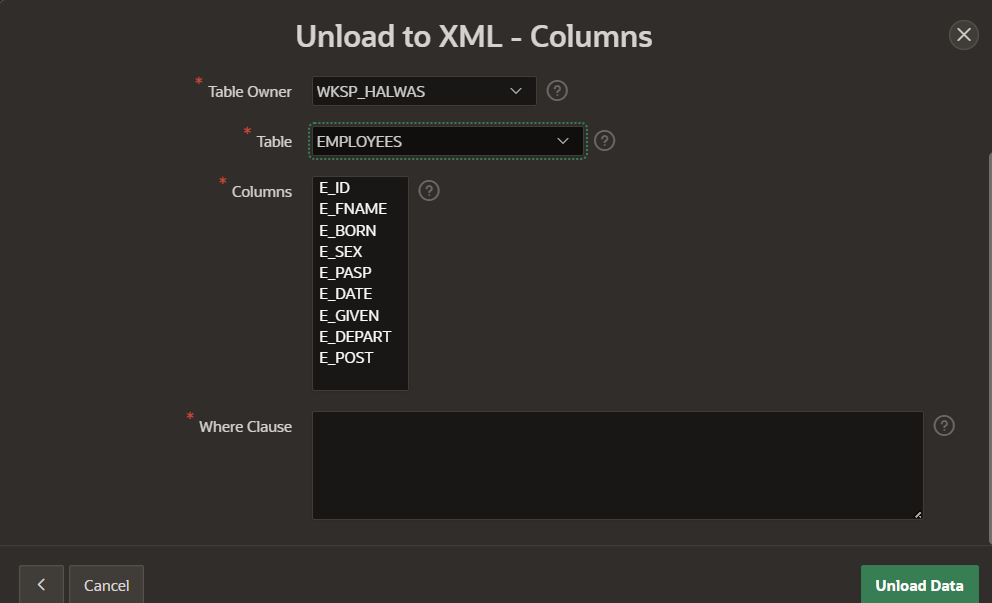


Рисунок 4.2 – Процесс экспорта таблицы базы данных

Удобно то, что можно экспортировать данные с определёнными условиями, используя поле Where Clause.

## **Экспорт данных**

При экспорте можно указывать дополнительные требования.

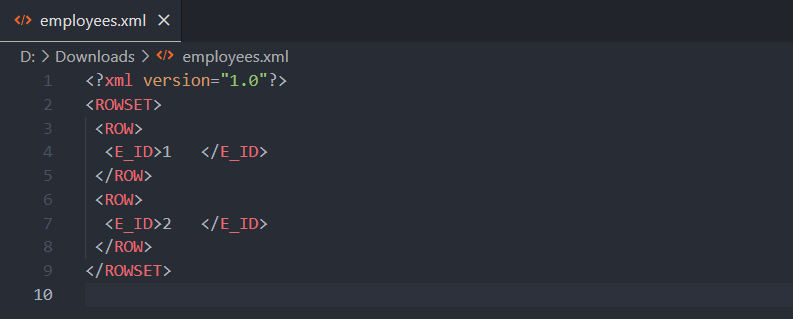


Рисунок 4.3 – Пример экспортированных данных в формате XML

Также, есть возможность и импортировать данные, имея необходимый файл:

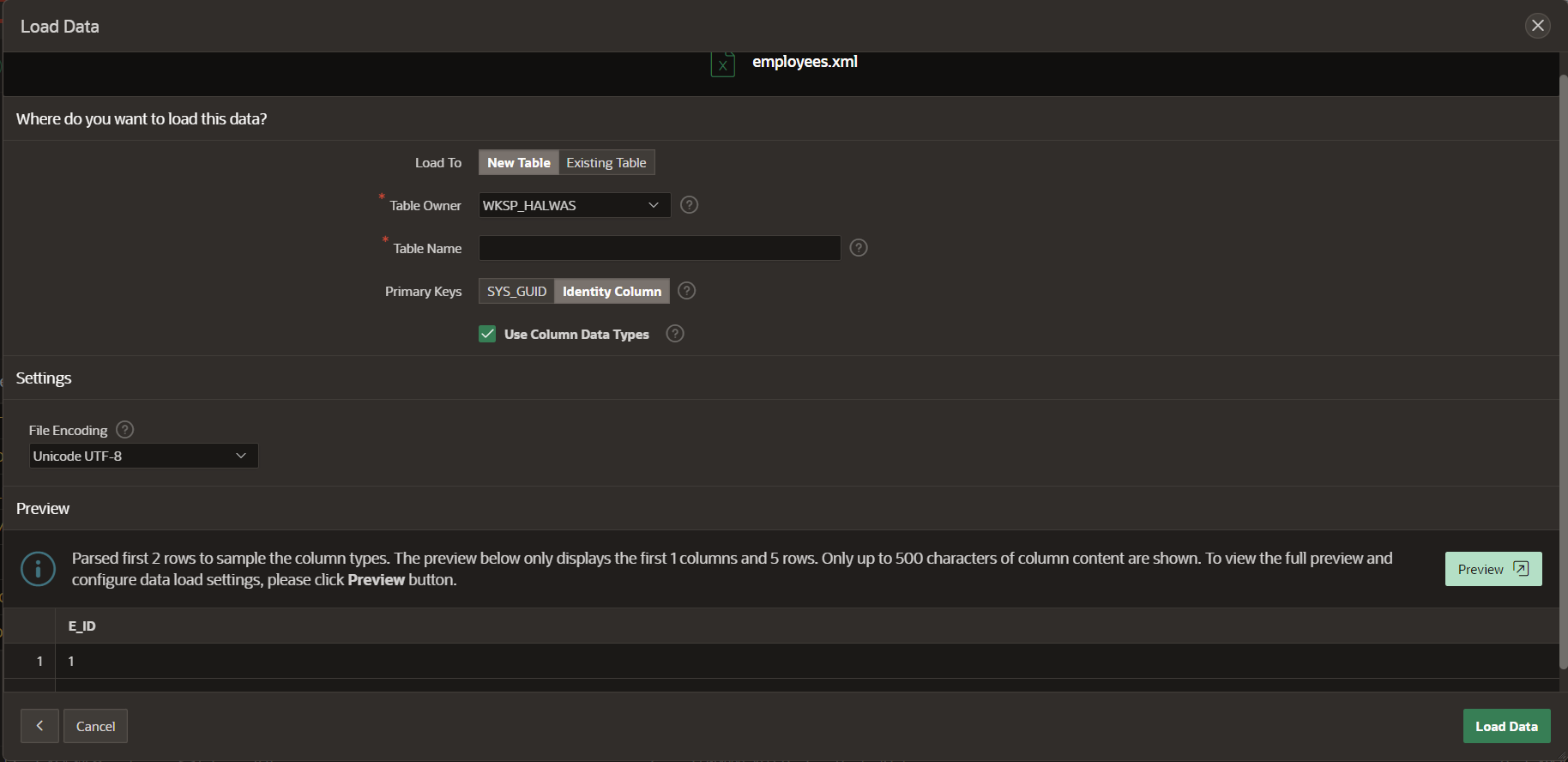


Рисунок 4.4 – Пример импорта данных

Стоит заметить, что с использованием такой технологии стоит быть предельно осторожным. Нужно следить за тем, какой пользователь базы данных имеет доступ к такой технологии, а также, когда и зачем он ей пользуется. Так, при несерьёзном подходе к данной ситуации владельцы рискуют потерять огромное количество информации.

# **Тестирование производительности**

## **Заполнение таблиц данными**

Для тестирования производительности базы данных есть разные способы. Но, для того чтобы что-то тестировать, нужно заполнить как минимум одну таблицу базы данных большим количеством строк. К примеру, заполним таблицу departs значениями, приблизительно, на 100000 строк:

|  |
| --- |
| DECLARE  l\_counter INTEGER := 30;  BEGIN  WHILE l\_counter < 100000 LOOP  l\_counter := l\_counter + 1;  INSERT INTO departs (d\_id, d\_name) VALUES (l\_counter, 'Department ' || l\_counter);  END LOOP;  END;  / |

Листинг 5.1 – Курсор для заполнения значениями таблицы departs

Проверим факт заполнения таблиц:

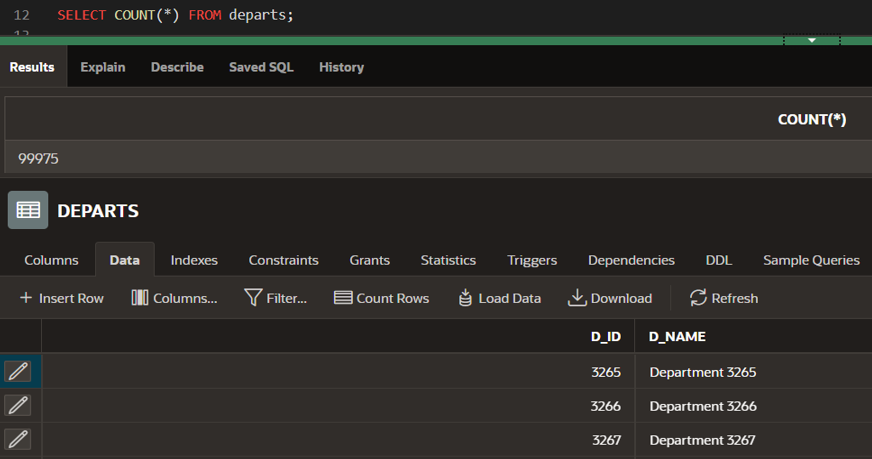


Рисунок 5.1 – Количество строк в таблице departs

Как видно, таблицы заполнены.

## **Создание дополнительных индексов**

Ускорить поиск по базе данных помогают индексы, которые, как правило, создаются по внешним ключам.

Для создания индексов, которые будут ускорять запросы с использованием операций JOIN и AND для таблиц departs, employees и orders, использовуем следующие SQL-запросы:

1. Воспользуемся ранее написанным индексом для таблицы employees

|  |
| --- |
| CREATE INDEX idx\_employees\_dept\_post ON employees (E\_DEPART, E\_POST); |

Листинг 5.2 – Индекс таблицы employees

1. Создадим новый индекс для таблицы orders:

|  |
| --- |
| CREATE INDEX idx\_orders\_dept\_cust ON orders (o\_depart, o\_customer); |

Листинг 5.3 – Индекс таблицы orders

После создания этих индексов можно вызвать запрос, который будет использовать их для ускорения операций JOIN и AND. Например, для получения списка всех отделов (departs), в которых работают сотрудники (employees) и которые заказывали товары (orders) у определенного клиента, можно использовать следующий SQL-запрос:

|  |
| --- |
| SELECT d.d\_id, d.d\_name  FROM departs d  JOIN employees e ON d.d\_id = e.E\_DEPART  JOIN orders o ON d.d\_id = o.o\_depart AND o.o\_customer = 123  ORDER BY d.d\_name; |

Листинг 5.4 – Запрос для тестирования производительности базы данных

Выполним данный запрос:

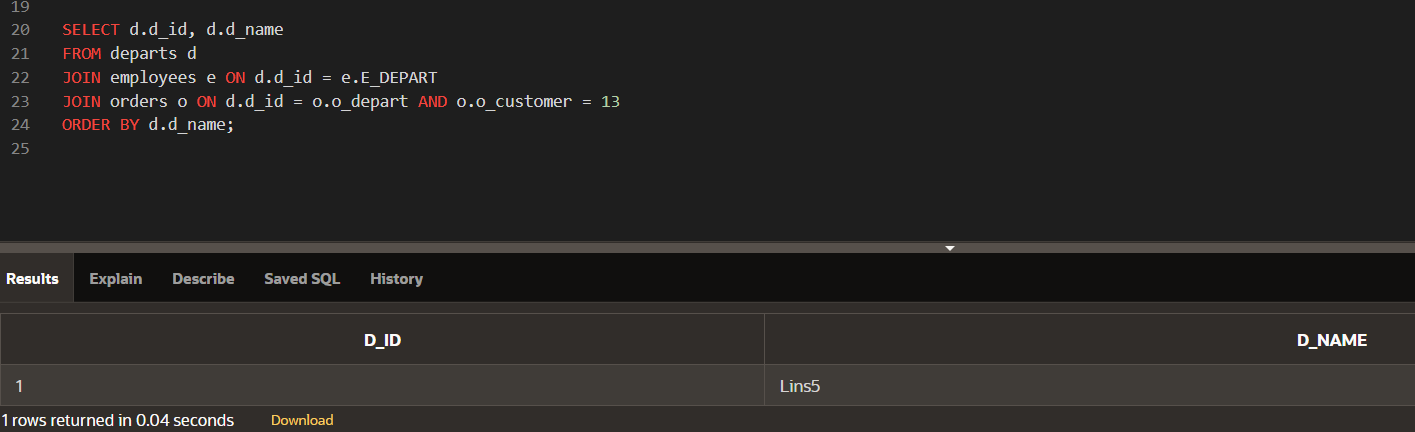


Рисунок 5.2 – Результат выполнения запроса

Как можно заметить, данный скрипт выполняется очень быстро – всего за 0.04 секунды, что является отличным результатом поиска информации среди такого большого количества записей.

# **Описание технологии и её применения в базе данных**

При разработке данного проекта была выбрана технология Oracle Apex. Полное название данной технологии – Oracle Application Express, ранее называемая как Oracle HTMLDB. Это проприетарная среда быстрой разработки прикладного программного обеспечения на основе СУБД Oracle Database, целиком реализованная как веб-приложение. Все элементы, возникающие в цикле разработки приложения в данной среде, хранятся непосредственно в инфраструктуре Oracle Database, тем самым обеспечивается совместная работа разработчиков и контроль версий без использования файлов и дополнительных [систем управления версиями](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8). В качестве веб-сервера для отображения страниц могут использоваться Apache HTTP Server.

Приложение, разрабатываемое в рамках данного курсового проекта, целиком реализуется на экземпляре Oracle Database. Всё это хранится на серверах Oracle. Достаточно завести аккаунт Apex и создать рабочее пространство, после чего можно выполнять все необходимые действия.

Все действия, выполненные с базой данных ранее, были произведены в веб-приложении Apex.

Перед тем, как использовать Apex, нужно авторизироваться в веб-приложении. Как раз на этом этапе и происходит дальнейший выбор пользователя приложения, будь то администратор, пользователь или разработчик.

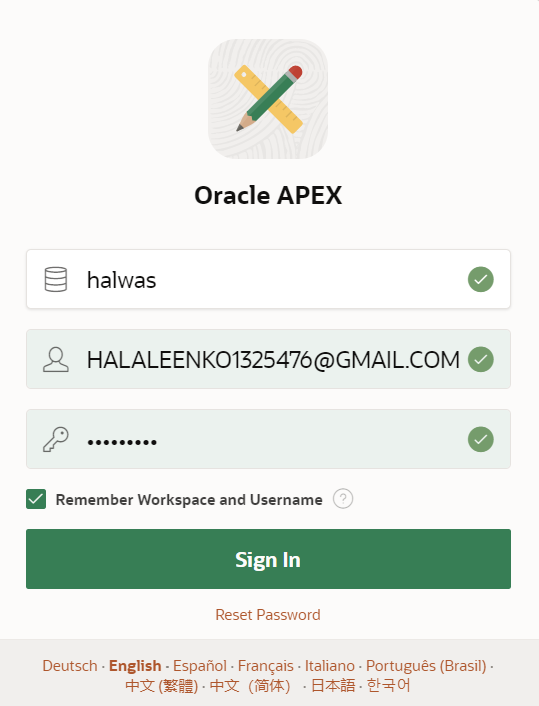


Рисунок 6.1 – Авторизация в Apex

Далее пользователей приветствуют 4 основные вкладки: App Builder для создания приложений, SQL Workshop для работы с базой данных, Team Development для командной разработки и Gallery для возможности взять готовый шаблон приложения либо что-то подобное.

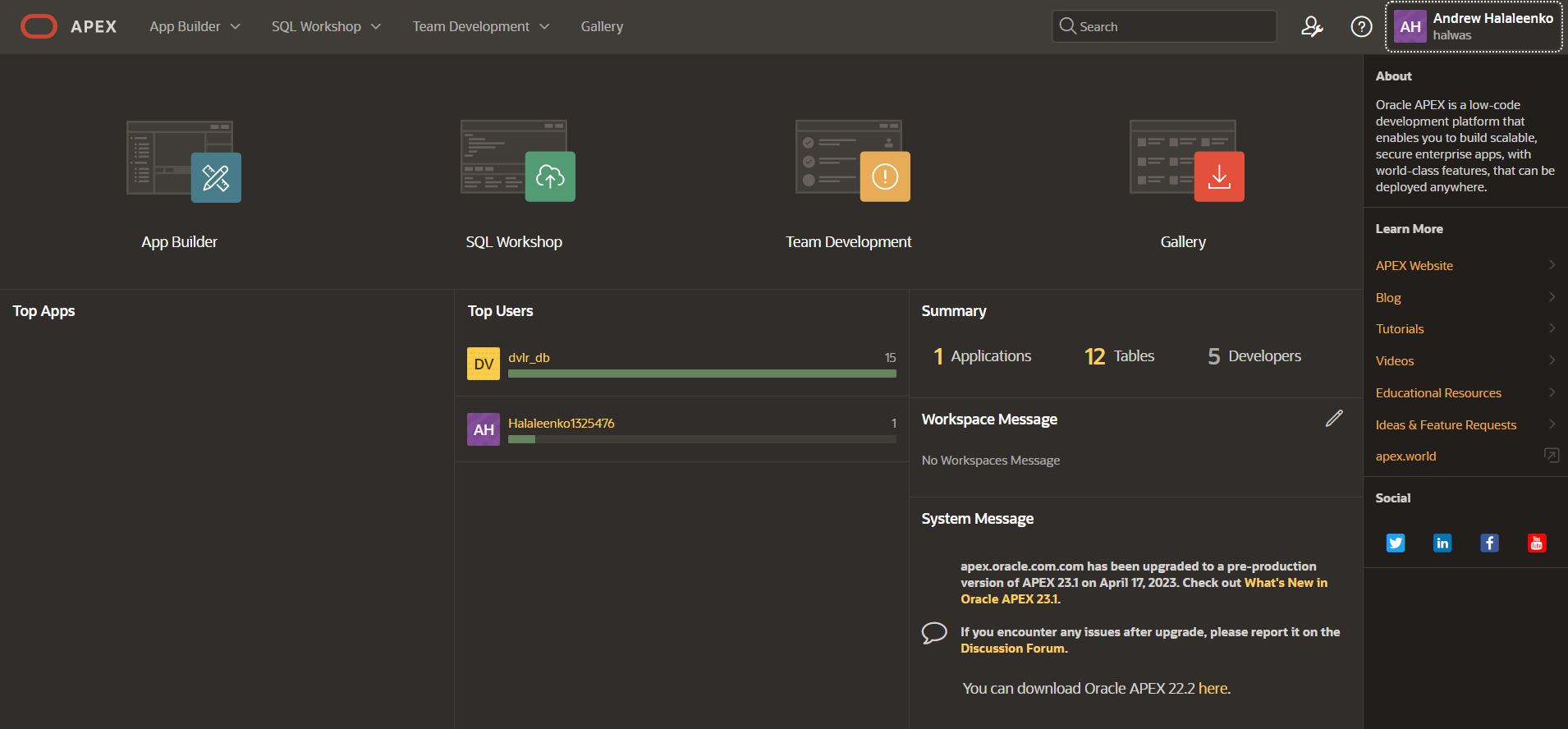


Рисунок 6.2 – Главный экран Apex

Вся предыдущая часть проекта была создана во вкладке SQL Workshop.

Данное веб-приложение предлагает просто огромное количество возможностей. Можно напрямую отслеживать любые активности пользователей сервиса, менять привилегии вести аналитику и так далее.

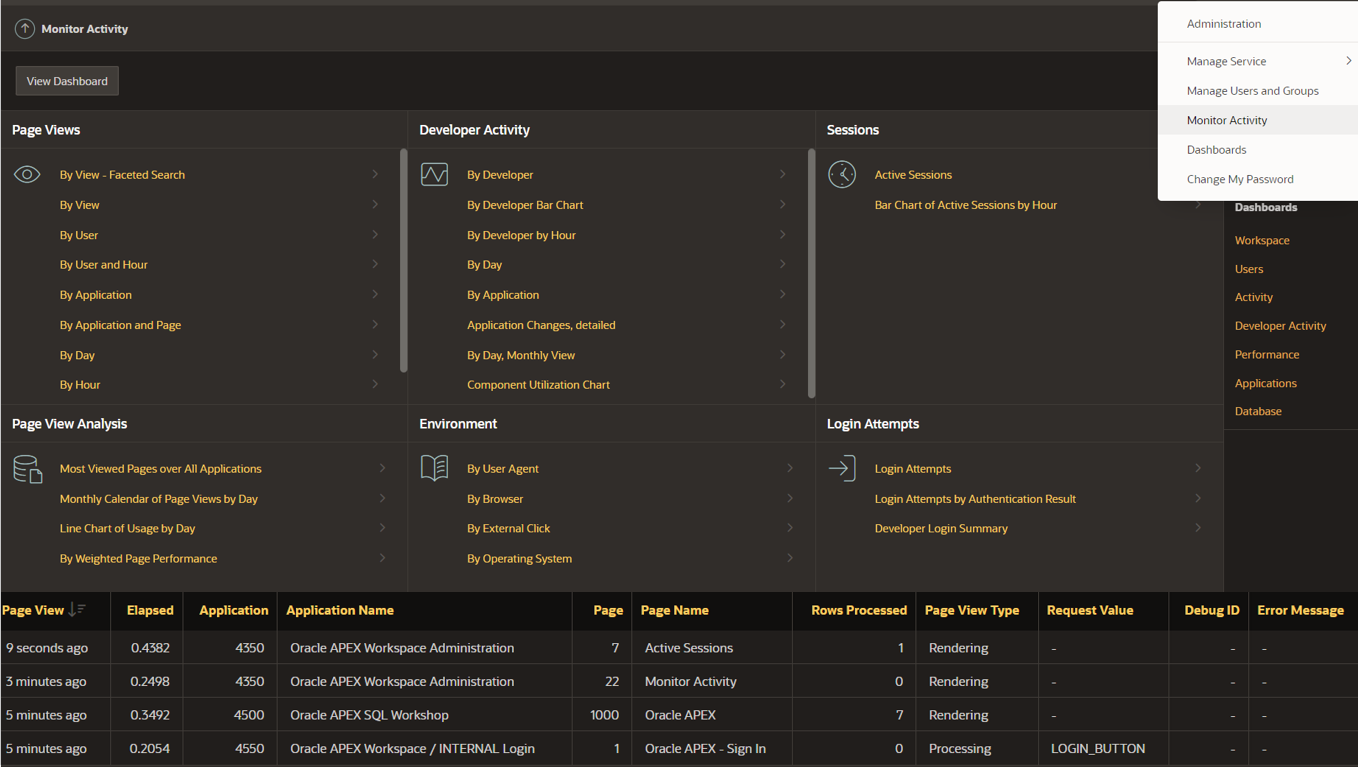


Рисунок 6.3 – Отслеживание активности в Apex

Конечно же, для начинающих пользователей эти возможности от Oracle будут чем-то новым и непонятным. Поэтому, конечно-же, Oracle предлагает ознакомится с официальной документацией данного продукта.

По итогу говоря, Oracle Apex предлагает множество интересных и полезных функций и возможностей. Основные из них:

* Interactive Grid – это мощный инструмент, который позволяет пользователю просматривать, редактировать и удалять данные в таблице, а также выполнять другие операции.
* Dynamic Actions - позволяет создавать динамические события, которые реагируют на различные действия пользователей, такие как нажатие кнопки или выбор элемента в списке.
* Plug-ins – это расширения для Oracle Apex, которые добавляют новые функциональные возможности в приложение, такие как графики, календари, деревья и т. д.
* Universal Theme – это тема, которая позволяет создавать современные и респонсивные веб-страницы с помощью множества различных компонентов и виджетов.
* Team Development – это функция, которая позволяет нескольким разработчикам работать над одним приложением одновременно.
* RESTful Services – это API, которое позволяет обмениваться данными между Oracle Apex и другими приложениями и сервисами, используя стандартные протоколы REST.
* Oracle JET – это набор инструментов и библиотек JavaScript, которые могут быть использованы в Oracle Apex для создания богатых и интерактивных пользовательских интерфейсов.
* Mobile Application Development – Oracle Apex позволяет создавать мобильные приложения для iOS и Android, используя нативные функции и возможности устройств.
* Oracle Machine Learning – это инструмент, который позволяет создавать и обучать модели машинного обучения прямо в Oracle Apex, что может помочь улучшить качество данных и принимаемых решений.
* Oracle Application Express (APEX) Office Print – это расширение, которое позволяет создавать и печатать отчеты в различных форматах (PDF, Excel, Word) прямо из приложения Oracle Apex.

# **Краткое описание приложения для демонстрации**

Создадим приложение для удобного использования нашей базы данных. Это также будет сделано с использованием технологии Oracle Apex. Теперь, для этого будет использоваться вкладка App Builder, доступ к которой имеется лишь у определённых разработчиков и администраторов. Поэтому, следует авторизироваться под пользователем, созданным в рамках данного проекта конкретно для разработки приложений.

Кликаем create, вводим название, задаём начальную страницу, выбираем необходимые параметры и создаём приложение для нашего сервиса. Очень удобной является возможность выбора языка приложения – поддерживается большое множество, в том числе и русский. При необходимости, все ненужные страницы можно отредактировать или удалить в любой момент времени.

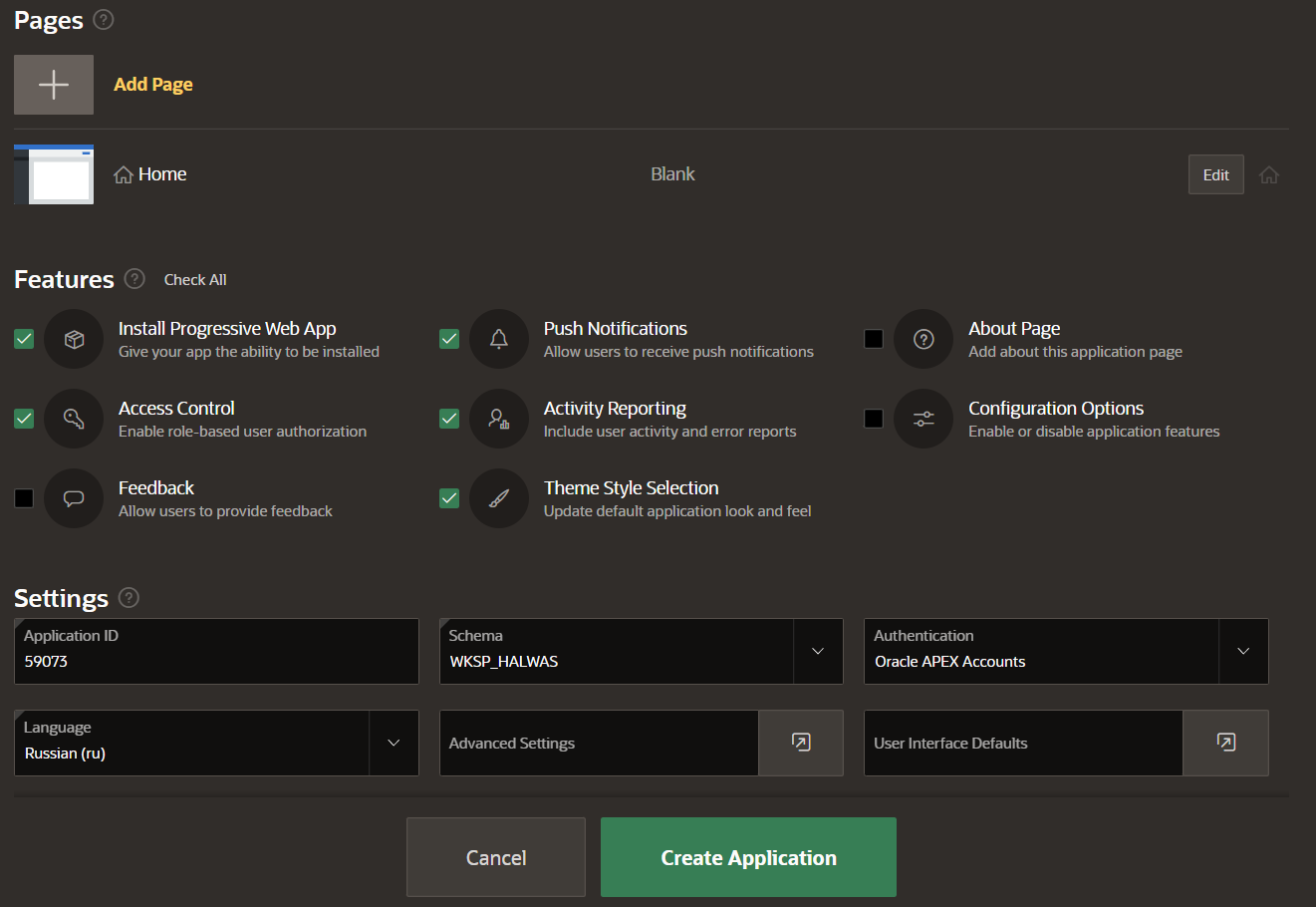


Рисунок 7.1 – Создание приложения

Создав приложение, можно сразу же увидеть большое количество уже сгенерированных при создании страниц, а также прочие вкладки и инструменты, которые помогут создать данное приложение.

На данный момент необходимо создать простое приложение, которое даст возможность просматривать информацию, хранящуюся в базе данных.

Вообще, Apex предлагает самые разнообразные способы представления данных. Можно выводить как просто текстовые значения, так и заниматься их динамическим анализом, преобразованием в графики и так далее.

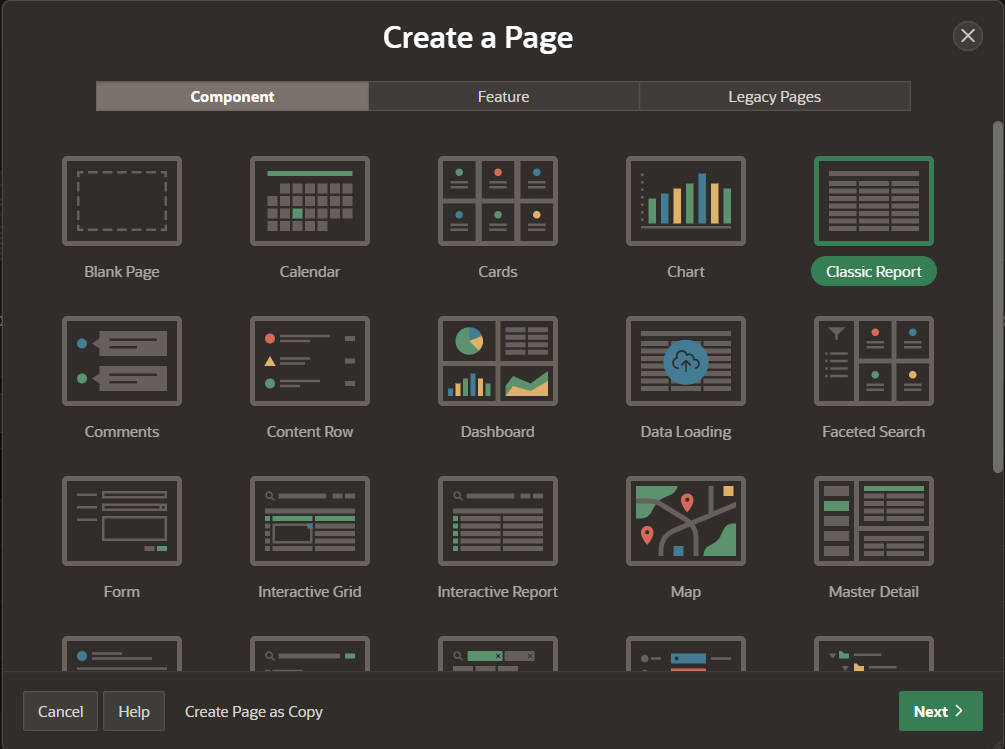


Рисунок 7.2 – Возможные страницы представления данных

Для простого представления данных в виде таблиц выбирается пункт Classic Report.

Если допускается ситуация, при которой случайно была создана страница с ошибкой, не та страница либо прочее, то можно удобно удалить её, используя элементы интерфейса:

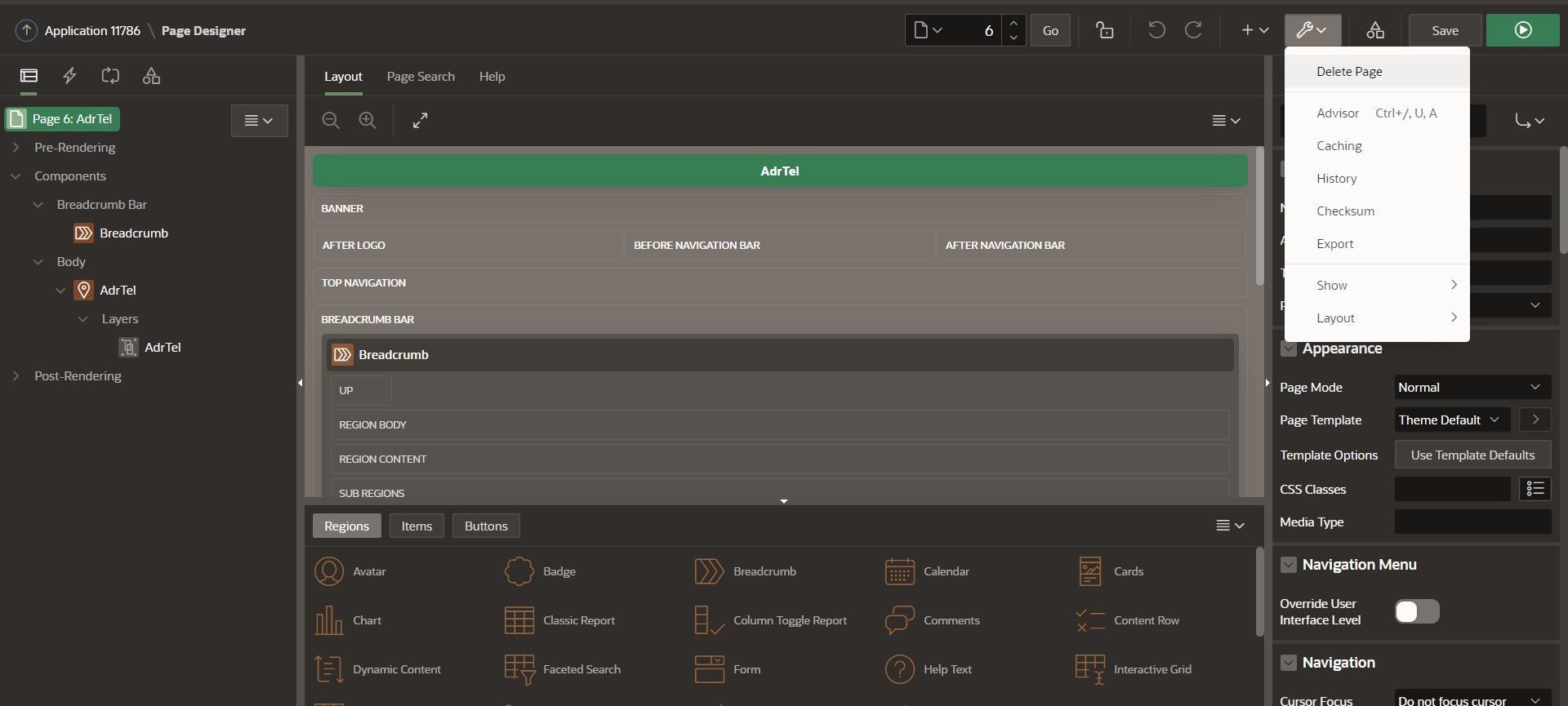


Рисунок 7.3 – Удаление страницы

Проделав аналогичные шаги для остальных страниц, привязывая к ним нужные страницы, получаем простое и удобное приложение для просмотра данных в табличном представлении:

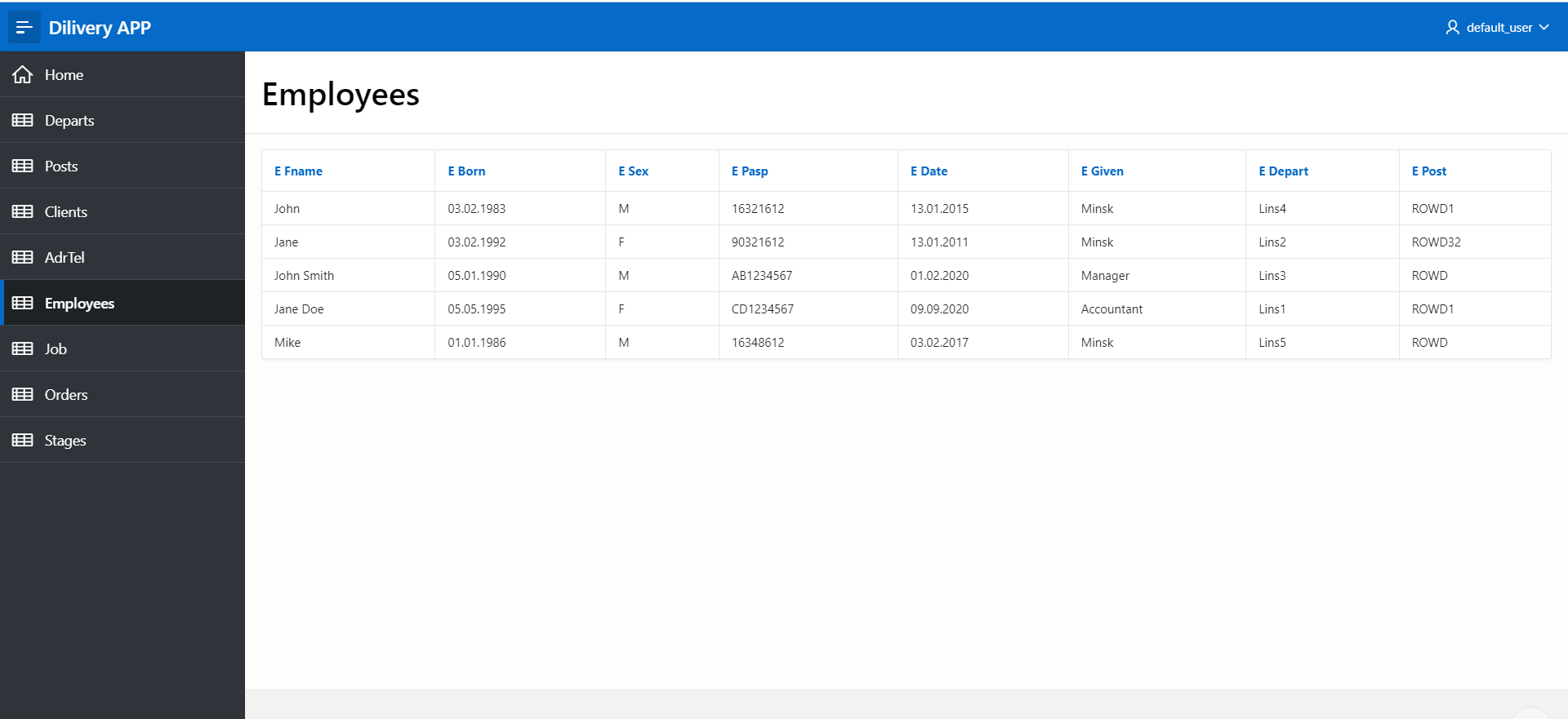


Рисунок 7.4 – Приложение просмотра данных

На данном этапе разработка приложения для представления данных будет завершена.

# **Руководство пользователя**

В данном приложении в роли пользователя выступают аналитики и прочие люди, занимающиеся изучением и анализом данных.

Первоначально, необходимо перейти на официальный сайт продукта и идентифицироваться под пользователем с логином default\_user и паролем halwa2003.

После этого пользователю открывается доступ для просмотра содержимого приложения, но не более. Он не может, примеру, заниматься редактированием данных и прочим – роль default\_user большего не позволяет.

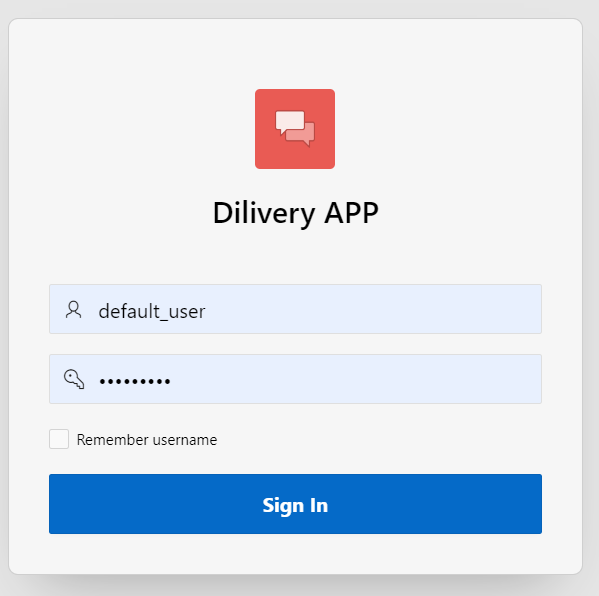


Рисунок 8.1 – Авторизация пользователя

После этого пользователь получает доступ к данным.

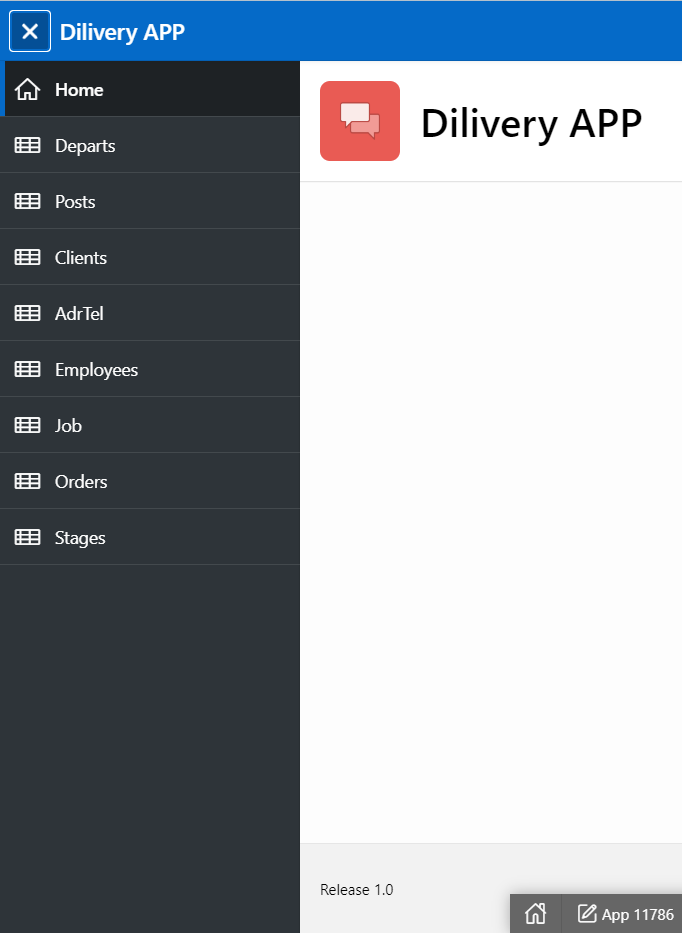


Рисунок 8.2 – Меню страниц с данными

В боковом меню справа пользователь может найти необходимые ему страницы с данными.

# **Заключение**

Разработка проекта подходит к концу. Сервисом можно пользоваться без каких-либо проблем и затруднений. Также, в будущем, проект может быть расширен как в направлении приложения, так и в направлении базы данных, так как стремиться ещё есть куда.

В ходе разработки сервиса были освоены некоторые возможности Oracle sql, которые раньше не были изучены. Было осуществлено знакомство с технологией Apex CMS, позволившей создать весь это проект за счёт своих внутренних возможностей.

Данный проект создавался со следующими целями:

* разработать базу данных для сервиса доставки;
* реализовать доступ к определённым данным через процедуры, функции и другие объекты базы данных;
* импортировать базу данных в файлы формата XML;
* протестировать производительность базы данных на примере заполнения её большим количеством данных и, при необходимости, реализовать дополнительные объекты базы данных, такие как индексы, для ускорения поиска данных;
* реализовать выбранную технологию базы данных;
* создать пользовательское приложение для представления данных.

После завершения работ над проектом можно заявить о том, что все намеченные цели были выполнены.

# **Список используемых источников**

1 Управление User и Group в Workspace, [Электронный ресурс]. – Режим доступа: https://betacode.net/10345/oracle-apex-tutorial-for-beginners#a361484. – Дата доступа: 24.04.2023. Пункт номер 4.

2 Статья по использованию Oracle Apex, [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/445128/ – Дата доступа: 05.04.2023.

3 Статья Oracle PL/SQL учебник, [Электронный ресурс]. – Режим доступа: https://oracleplsql.ru/contents-oracle-plsql.html. – Дата доступа: 06.04.2023.

# **Приложение А Листинг кода для создания таблиц**

|  |
| --- |
| CREATE TABLE departs (  d\_id int PRIMARY KEY,  d\_name VARCHAR2(100)  );  create table posts(  p\_posts varchar2(30) PRIMARY KEY,  p\_sal numeric(8,2) not null,  check (p\_sal > 450)  );  create table employees(  E\_ID int PRIMARY KEY,  E\_fname varchar(25) not null,  E\_BORN date not null,  E\_SEX char(1) not null,  E\_PASP char(10) unique,  E\_DATE date not null,  E\_GIVEN varchar2(50) not null,  E\_DEPART int references departs (d\_id),  E\_POST varchar2(30) references posts (p\_posts)  );  create table adrtel (  A\_ID int references employees (E\_ID),  A\_ADDR varchar2(50),  A\_PHONE varchar2(30)  );  create table clients (  C\_ID int PRIMARY KEY,  C\_CUSTOMER varchar2(40) not null,  C\_ADR varchar2(50) not null,  C\_PERSON varchar2(50) not null,  C\_PHONE varchar2(30)  );  create table orders (  o\_abbr char(10) primary key,  o\_title varchar2(100) not null,  o\_depart int references departs (d\_id),  o\_customer int references clients (c\_id),  o\_begin date not null,  o\_end date not null,  check (o\_end > o\_begin), --дата окончания заказа больше даты --начала  o\_cost nchar(10) not null,  check (o\_cost > 0)  );  create table stages (  s\_ord char(10) references orders (o\_abbr),  s\_num numeric(5),  s\_title varchar2(100) not null,  s\_begin date not null,  s\_end date not null,  check (s\_end > s\_begin),  s\_cost nchar(10) not null,  s\_sum nchar (10) not null  );  create table job (  J\_PRO char(10) references orders (o\_abbr),  J\_EMP int references employees (E\_ID),  J\_ROLE varchar2(20) not null,  J\_BONUS numeric(2)  ); |

# **Приложение Б Листинг кода процедур для работы с базой данных**

|  |
| --- |
| --Процедура добавления новой работы  CREATE OR REPLACE PROCEDURE add\_job (  p\_order\_abbr IN orders.o\_abbr%TYPE,  p\_emp\_id IN employees.e\_id%TYPE,  p\_role IN job.j\_role%TYPE,  p\_bonus IN job.j\_bonus%TYPE DEFAULT NULL  )  AS  BEGIN  INSERT INTO job (j\_pro, j\_emp, j\_role, j\_bonus)  VALUES (p\_order\_abbr, p\_emp\_id, p\_role, p\_bonus);  COMMIT;  END;  /  --Процедура обновления информации о работе  CREATE OR REPLACE PROCEDURE update\_job (  p\_order\_abbr IN orders.o\_abbr%TYPE,  p\_emp\_id IN employees.e\_id%TYPE,  p\_role IN job.j\_role%TYPE,  p\_bonus IN job.j\_bonus%TYPE DEFAULT NULL  )  AS  BEGIN  UPDATE job  SET j\_role = p\_role,  j\_bonus = p\_bonus  WHERE j\_pro = p\_order\_abbr AND j\_emp = p\_emp\_id;  COMMIT;  END;  /  --Процедура обновления информации о работе  CREATE OR REPLACE PROCEDURE update\_job (  p\_order\_abbr IN orders.o\_abbr%TYPE,  p\_emp\_id IN employees.e\_id%TYPE,  p\_role IN job.j\_role%TYPE,  p\_bonus IN job.j\_bonus%TYPE DEFAULT NULL  )  AS  BEGIN  UPDATE job  SET j\_role = p\_role,  j\_bonus = p\_bonus  WHERE j\_pro = p\_order\_abbr AND j\_emp = p\_emp\_id;  COMMIT;  END;  /  --Процедура удаления работы  CREATE OR REPLACE PROCEDURE delete\_job (  p\_order\_abbr IN orders.o\_abbr%TYPE,  p\_emp\_id IN employees.e\_id%TYPE  )  AS  BEGIN  DELETE FROM job  WHERE j\_pro = p\_order\_abbr AND j\_emp = p\_emp\_id;  COMMIT;  END;  /  -- Вставка записи в таблицу departs  CREATE OR REPLACE PROCEDURE insert\_depart(  p\_id IN departs.d\_id%TYPE,  p\_name IN departs.d\_name%TYPE  )  AS  BEGIN  INSERT INTO departs (d\_id, d\_name) VALUES (p\_id, p\_name);  COMMIT;  END;  /  -- Обновление записи в таблице departs  CREATE OR REPLACE PROCEDURE update\_depart(  p\_id IN departs.d\_id%TYPE,  p\_name IN departs.d\_name%TYPE  )  AS  BEGIN  UPDATE departs SET d\_name = p\_name WHERE d\_id = p\_id;  COMMIT;  END;  /  -- Удаление записи из таблицы departs  CREATE OR REPLACE PROCEDURE delete\_depart(p\_id IN departs.d\_id%TYPE)  AS  BEGIN  DELETE FROM departs WHERE d\_id = p\_id;  COMMIT;  END;  /  -- Получение списка всех записей из таблицы departs  CREATE OR REPLACE PROCEDURE get\_all\_departs(p\_cursor OUT SYS\_REFCURSOR)  AS  BEGIN  OPEN p\_cursor FOR SELECT \* FROM departs;  END;  /  -- Получение записи из таблицы departs по id  CREATE OR REPLACE PROCEDURE get\_depart\_by\_id(  p\_id IN departs.d\_id%TYPE,  p\_cursor OUT SYS\_REFCURSOR  )  AS  BEGIN  OPEN p\_cursor FOR SELECT \* FROM departs WHERE d\_id = p\_id;  END;  /  -- Вставка записи в таблицу posts  CREATE OR REPLACE PROCEDURE insert\_post(  p\_posts IN posts.p\_posts%TYPE,  p\_sal IN posts.p\_sal%TYPE  )  AS  BEGIN  INSERT INTO posts (p\_posts, p\_sal) VALUES (p\_posts, p\_sal);  COMMIT;  END;  /  -- Обновление записи в таблице posts  CREATE OR REPLACE PROCEDURE update\_post(  p\_posts IN posts.p\_posts%TYPE,  p\_sal IN posts.p\_sal%TYPE  )  AS  BEGIN  UPDATE posts SET p\_sal = p\_sal WHERE p\_posts = p\_posts;  COMMIT;  END;  /  -- Удаление записи из таблицы posts  CREATE OR REPLACE PROCEDURE delete\_post(p\_posts IN posts.p\_posts%TYPE)  AS  BEGIN  DELETE FROM posts WHERE p\_posts = p\_posts;  COMMIT;  END;  /  -- Получение списка всех записей из таблицы posts  CREATE OR REPLACE PROCEDURE get\_all\_posts(p\_cursor OUT SYS\_REFCURSOR)  AS  BEGIN  OPEN p\_cursor FOR SELECT \* FROM posts;  END;  -- Получение записи из таблицы posts по p\_posts  CREATE OR REPLACE PROCEDURE get\_post\_by\_p\_posts(  p\_posts IN posts.p\_posts%TYPE,  p\_cursor OUT SYS\_REFCURSOR  ) AS  BEGIN  OPEN p\_cursor FOR SELECT \* FROM posts WHERE p\_posts = p\_posts;  END;  /  -- вставка записи в таблицу clients  CREATE OR REPLACE PROCEDURE add\_client (  p\_C\_ID IN clients.C\_ID%TYPE,  p\_CUSTOMER IN clients.C\_CUSTOMER%TYPE,  p\_ADR IN clients.C\_ADR%TYPE,  p\_PERSON IN clients.C\_PERSON%TYPE,  p\_PHONE IN clients.C\_PHONE%TYPE  )AS  BEGIN  INSERT INTO clients (C\_ID, C\_CUSTOMER, C\_ADR, C\_PERSON, C\_PHONE)  VALUES (p\_C\_ID, p\_CUSTOMER, p\_ADR, p\_PERSON, p\_PHONE);  COMMIT;  END add\_client;  /  -- обновление записи в таблице clients  CREATE OR REPLACE PROCEDURE update\_client (  p\_C\_ID IN clients.C\_ID%TYPE,  p\_CUSTOMER IN clients.C\_CUSTOMER%TYPE,  p\_ADR IN clients.C\_ADR%TYPE,  p\_PERSON IN clients.C\_PERSON%TYPE,  p\_PHONE IN clients.C\_PHONE%TYPE  ) AS  BEGIN  UPDATE clients  SET C\_CUSTOMER = p\_CUSTOMER,  C\_ADR = p\_ADR,  C\_PERSON = p\_PERSON,  C\_PHONE = p\_PHONE  WHERE C\_ID = p\_C\_ID;  COMMIT;  END update\_client;  /  -- удаление записи из таблицы clients  CREATE OR REPLACE PROCEDURE delete\_client (  p\_C\_ID IN clients.C\_ID%TYPE  )  AS  BEGIN  DELETE FROM clients WHERE C\_ID = p\_C\_ID;  COMMIT;  END delete\_client;  /  -- Процедура для добавления записи в таблицу orders  CREATE OR REPLACE PROCEDURE add\_order(  abbr IN CHAR,  title IN VARCHAR2,  depart IN INT,  customer IN INT,  begin\_date IN DATE,  end\_date IN DATE,  cost IN NCHAR  )  AS  BEGIN  INSERT INTO orders(o\_abbr, o\_title, o\_depart, o\_customer, o\_begin, o\_end, o\_cost)  VALUES(abbr, title, depart, customer, begin\_date, end\_date, cost);  END;  /  -- Процедура для обновления записи в таблице orders  CREATE OR REPLACE PROCEDURE update\_order(  abbr IN CHAR,  title IN VARCHAR2,  depart IN INT,  customer IN INT,  begin\_date IN DATE,  end\_date IN DATE,  cost IN NCHAR  )  AS  BEGIN  UPDATE orders  SET o\_title = title, o\_depart = depart, o\_customer = customer, o\_begin = begin\_date, o\_end = end\_date, o\_cost = cost  WHERE o\_abbr = abbr;  END;  /  -- Процедура для удаления записи из таблицы orders  CREATE OR REPLACE PROCEDURE delete\_order(  abbr IN CHAR  )  AS  BEGIN  DELETE FROM orders  WHERE o\_abbr = abbr;  END;  /  -- Процедура для получения списка всех записей в таблице orders  CREATE OR REPLACE PROCEDURE get\_orders(  orders\_list OUT SYS\_REFCURSOR  )AS  BEGIN  OPEN orders\_list FOR  SELECT \*  FROM orders;  END;  /  --Процедура для получения записи из таблицы orders по ее  --идентификатору  CREATE OR REPLACE PROCEDURE get\_order(  abbr IN CHAR,  order\_data OUT SYS\_REFCURSOR  )  AS  BEGIN  OPEN order\_data FOR  SELECT \*  FROM orders  WHERE o\_abbr = abbr;  END;  /  -- Процедура для получения общей стоимости заказов по каждому клиенту --из таблицы orders  CREATE OR REPLACE PROCEDURE get\_total\_cost\_by\_customer(  total\_cost OUT SYS\_REFCURSOR  )  AS  BEGIN  OPEN total\_cost FOR  SELECT o\_customer, SUM(o\_cost) as total\_cost  FROM orders  GROUP BY o\_customer;  END;  /  -- Процедура для получения общей стоимости заказов по каждому отделу --из таблицы orders  CREATE OR REPLACE PROCEDURE get\_total\_cost\_by\_depart(  total\_cost OUT SYS\_REFCURSOR  )AS  BEGIN  OPEN total\_cost FOR  SELECT o\_depart, SUM(o\_cost) as total\_cost  FROM orders  GROUP BY o\_depart;  END;  /  --Процедура для получения списка заказов, начинающихся с определенной --даты  CREATE OR REPLACE PROCEDURE get\_orders\_starting\_from(  start\_date IN DATE,  orders\_list OUT SYS\_REFCURSOR  )AS  BEGIN  OPEN orders\_list FOR  SELECT \*  FROM orders  WHERE o\_begin >= start\_date;  END;  /  --Процедура для добавления записи в таблицу  CREATE OR REPLACE PROCEDURE insert\_stage (  p\_ord IN CHAR,  p\_num IN NUMBER,  p\_title IN VARCHAR2,  p\_begin IN DATE,  p\_end IN DATE,  p\_cost IN CHAR,  p\_sum IN CHAR  )  IS  BEGIN  INSERT INTO stages (s\_ord, s\_num, s\_title, s\_begin, s\_end, s\_cost, s\_sum)  VALUES (p\_ord, p\_num, p\_title, p\_begin, p\_end, p\_cost, p\_sum);  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Stage ' || p\_title || ' added successfully');  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  END;  /  --Процедура для обновления записи в таблице  CREATE OR REPLACE PROCEDURE update\_stage (  p\_ord IN CHAR,  p\_num IN NUMBER,  p\_title IN VARCHAR2,  p\_begin IN DATE,  p\_end IN DATE,  p\_cost IN CHAR,  p\_sum IN CHAR  )  IS  BEGIN  UPDATE stages  SET s\_title = p\_title,  s\_begin = p\_begin,  s\_end = p\_end,  s\_cost = p\_cost,  s\_sum = p\_sum  WHERE s\_ord = p\_ord AND s\_num = p\_num;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Stage ' || p\_title || ' updated successfully');  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  END;  /  --Процедура для удаления записи из таблицы  CREATE OR REPLACE PROCEDURE delete\_stage (  p\_ord IN CHAR,  p\_num IN NUMBER  )  IS  BEGIN  DELETE FROM stages  WHERE s\_ord = p\_ord AND s\_num = p\_num;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Stage with order ' || p\_ord || ' and number ' || p\_num || ' deleted successfully');  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  END;  /  --Процедура для получения списка всех записей из таблицы  CREATE OR REPLACE PROCEDURE get\_all\_stages IS  BEGIN  FOR stage IN (SELECT \* FROM stages)  LOOP  DBMS\_OUTPUT.PUT\_LINE(stage.s\_ord || ' ' || stage.s\_num || ' ' || stage.s\_title || ' ' || stage.s\_begin || ' ' || stage.s\_end || ' ' || stage.s\_cost || ' ' || stage.s\_sum);  END LOOP;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  END;  / |