**Реферат**

Пояснительная записка курсового проекта содержит 27 страниц пояснительной записки, иллюстраций, источников литературы, приложения.

ORACLE 19c, .NET FRAMEWORK, ORACLE DEVELOPER TOOLS, ODT.NET, RMAN.

Основная цель курсового проекта: проектирование базы данных для приложения по оказанию сервиса ремонтных услуг и освоение технологии восстановления данных.

В первой главе проводится аналитический обзор литературы по тематике курсового проекта и содержит анализ методов решения поставленных задач.

Вторая глава посвящена процессу проектирования системы.

В третьей главе приведено обоснование технических приёмов.

В четвертой главе описан процесс тестирования, позволяющий понять интерфейс программного средства и предоставить доказательства работоспособности приложения.

В заключении приведены результаты проделанной работы.

**Содержание**

[Введение 3](#_Toc122955523)

[1 Аналитический обзор литературы 4](#_Toc122955524)

[1.1 Аналитический обзор источников 4](#_Toc122955525)

[1.2 Обзор аналогов 4](#_Toc122955526)

[1. Разработка базы данных 6](#_Toc122955527)

[2.1 Проектирование базы данных 6](#_Toc122955528)

[1.2 Процедуры для решения поставленных задач 7](#_Toc122955529)

[1.3 Индексы 7](#_Toc122955530)

[1.4 Создание пользователей и ролей 8](#_Toc122955531)

[2. Технология маскирования и шифрования в БД 10](#_Toc122955532)

[3. Импорт и экспорт данных 12](#_Toc122955533)

[4. Тестирование производительности 14](#_Toc122955534)

[Заключение 15](#_Toc122955535)

[Список использованных источников 16](#_Toc122955536)

[Приложение А 17](#_Toc122955537)

[Диаграмма базы данных 17](#_Toc122955538)

[Приложение Б 18](#_Toc122955539)

[Процедуры авторизации и регистрации пользователя 18](#_Toc122955540)

[Процедуры для вывода данных 19](#_Toc122955541)

[Процедуры для покупки билета 21](#_Toc122955542)

[Процедуры для создания 22](#_Toc122955543)

[Процедуры для удаления 25](#_Toc122955544)

[Приложение В 26](#_Toc122955545)

[Процедура экспорта данных в XML 26](#_Toc122955546)

[Приложение Г 27](#_Toc122955547)

# Введение

Целью данной работы является разработка реляционной базы данных на тему «Сервис по ремонту техники». База данных должна быть составлена для использования в целях администрирования сервиса по ремонту, а именно: добавленных в список доступных к ремонту устройств, добавления новых мастеров и сбор статистики о их работе, а также для взаимодействия с клиентом посредством сбора и использования его личных данных. Также было необходимо разработать соответствующее приложение, для демонстрации её работы.

База данных — это организованная структура, предназначенная для хранения информации, систематизированная таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины.

Реляционная база данных — база данных, основанная на реляционной модели данных.

В качестве СУБД для базы данных была выбрана Oracle Database 19c, в связи с ее высокой производительностью, большим набором возможностей, широкой распространённостью и надежностью.

Необходимо создать программу для демонстрации работы базы данных. Было разработано приложение. Для входа в приложение необходимо пройти процедуру регистрации. После успешной регистрации пользователь имеет возможность войти в приложение под своим логином и введя пароль от своего аккаунта.

# Аналитический обзор литературы

## Аналитический обзор источников

В ходе подготовки пояснительной записки была изучена специальная техническая, учебно-методическая и справочная литература, статьи и материалы, опубликованные в сети интернет.

Основная информация о работе с средой разработки Visual Studio была получена из официальной документации

Принцип работы с SQL были получены из статьи «Подключение к базе данных». В статье было рассмотрено подключение необходимых библиотек, работа с SQL.

Концепция создания отчетов о питании пользователя была получена из статьи «Report-Driven Design». В статье была рассмотрена общая информация о создании отчёта и пример создания отчёта.

## Обзор аналогов

Для создания принципиально нового решения в виде программного продукта, используемого для контроля рациона пользователя, необходимо проанализировать уже существующее программные средства в данной сфере. Анализ достоинств и недостатков этих аналогов позволит сформировать требования к проектируемому программному средству, учитывающие опыт существующих разработок и внести в них улучшения или изменения.

В качестве исследуемых аналогов были выбраны программные продукты, которые можно отнести к группе приложений, предназначенных для заказа услуг, как наиболее близкие по области применения к разрабатываемому программному средству. Источником информации послужили электронные базы в сети Интернет.

В результате поиска были обнаружены следующие ресурсы:

* «Handsharker» – это программа, которая изначально создавалась для работы с различными сервисами, оказывающими услуги. Но в дальнейшем в неё был внедрён целый комплекс функций для взаимодействия между клиентом и предпринимателем. Функционал данного программного средства включает в себя:

1. Поиск услуги с помощью «строки поиска», отбор выполняется по базе услуг «Handsharker»;
2. Просмотр рекомендации других пользователей при выборе компаний, предоставляющих услуги;
3. Просмотр оценок и отзывов о конкретной услуге;
4. Добавление пользователя в качестве исполнителя услуги, с опциями предоставления информации о времени работы, прейскуранте и документы о государственной регистрации индивидуального предпринимателя;
5. Функция отправления сообщения на электронную почту исполнителя услуги.

* Мобильное приложение «Profi» – функциональный инструмент, позволяющий клиенту заказывать услуги. Приложение позволяет заказать не только услугу у конкретного исполнителя, но и оставить заявку, чтобы специалист сам мог ответить на неё. Что очень полезно людям, которым важно специалиста по более конкретным критериям. Также есть функция оценки и отзыва. Сами специалисты делятся на категории, и «Profi» ведет статистику того или иного вида деятельности.

# Разработка базы данных

* 1. Проектирование базы данных

При разработке курсового проекта понадобились следующие объекты:

1. Таблицы;
2. Хранимые процедуры;
3. Пользователи
4. Роли
5. Индексы

Для начала рассмотрим таблицы. Их диаграмма представлена на рисунке 2.1, а также в приложении А.

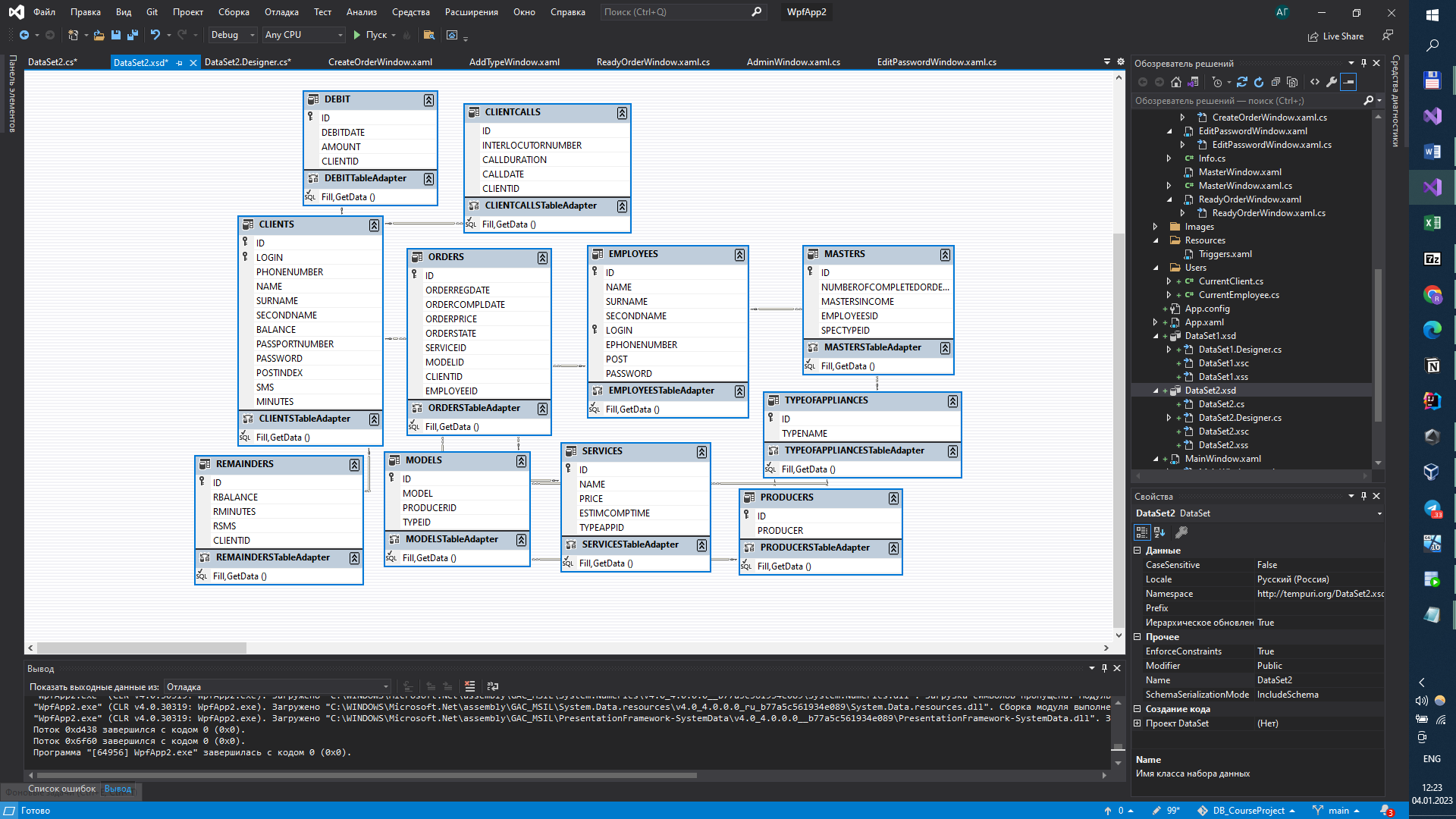


Рисунок 2.1 – Диаграмма таблиц пользовательских данных

База данных приложения состоит из 5 таблиц, связанных между собой внешними ключами.

Таблица «*Employees*» хранит в себе всю информацию о работниках, включая ФИО, логин, пароль, номер телефона, а также роль работника.

Таблица «*Clients*» хранит в себе всю информацию о клиентах, включая ФИО, баланс лицевого счёта, номер паспорта, почтовый индекс, количество бесплатных смс и минут общения с внутри сервиса и номер телефона, а также логин и пароль.

Таблица «*Orders*» хранит информацию о дате регистрации и окончании заказа, модели техники, цену и состояние заказа.

Таблица «*ClientCalls*» хранит в себе всю информацию о звонках клиента, включая набираемый номер, длительность звонка и дату звонка.

Таблица «*Models*» хранит всю информацию о единицах техники, фирму-производителя и тип техники.

Таблица «*Services*» хранит всю информацию о услугах, предоставляемых для определённых типов техники, название услуги, цену услуги и тип техники, для которого предоставляется услуга.

Таблица «*TypesOfAppliances*» содержит информацию о всех видах техники, обслуживаемых сервисом, а именно их названия и идентификаторы.

Таблица «*Masters*» хранит всю информацию о мастерах по ремонту, сумму общей прибыли, количество выполненных заказов и тип техники, с которым они работают.

Таблица «*Producers*» содержит информацию о всех производителях техники, модели которых обслуживаются сервисом, а именно их названия и идентификаторы.

Таблица «Debit» содержит информацию о всех пополнениях счета клиента.

Таблица «*Remainders*» хранит всю информацию о оставшихся на личном счёте клиента минут, доступных для звонка и сообщений.

* 1. Процедуры для решения поставленных задач

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

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

1. Выборка данных из таблиц;
2. Выборка данных по поисковому запросу;
3. Заполнение таблиц 100 000 строк;
4. Вход в аккаунт;
5. Удаление данных из таблиц;
6. Добавление данных в таблицы;
7. Экспорт и импорт таблицы ServicePack в формате xml;

Процедуры представлены в приложении Б.

* 1. Индексы

Для оптимизации работы с поисковыми запросами, необходимо использовать индексы. Так, для работы с наиболее часто используемыми таблицами, были разработаны наборы индексов по наиболее используемым выборкам. Для примера в листинге 2.2 приведено создание индекса SERVICEPACKIND.

create index SERVICESIND on SERVICES(Name);

Листинг 2.1 – Индекс SERVICESIND

* 1. Создание пользователей и ролей

Для базы данных было разработано два типа пользователей. Один из них разработан в качестве клиента сервиса – client, а второй для администрирования в базе данных и работы с приложением в качестве работника – employee.

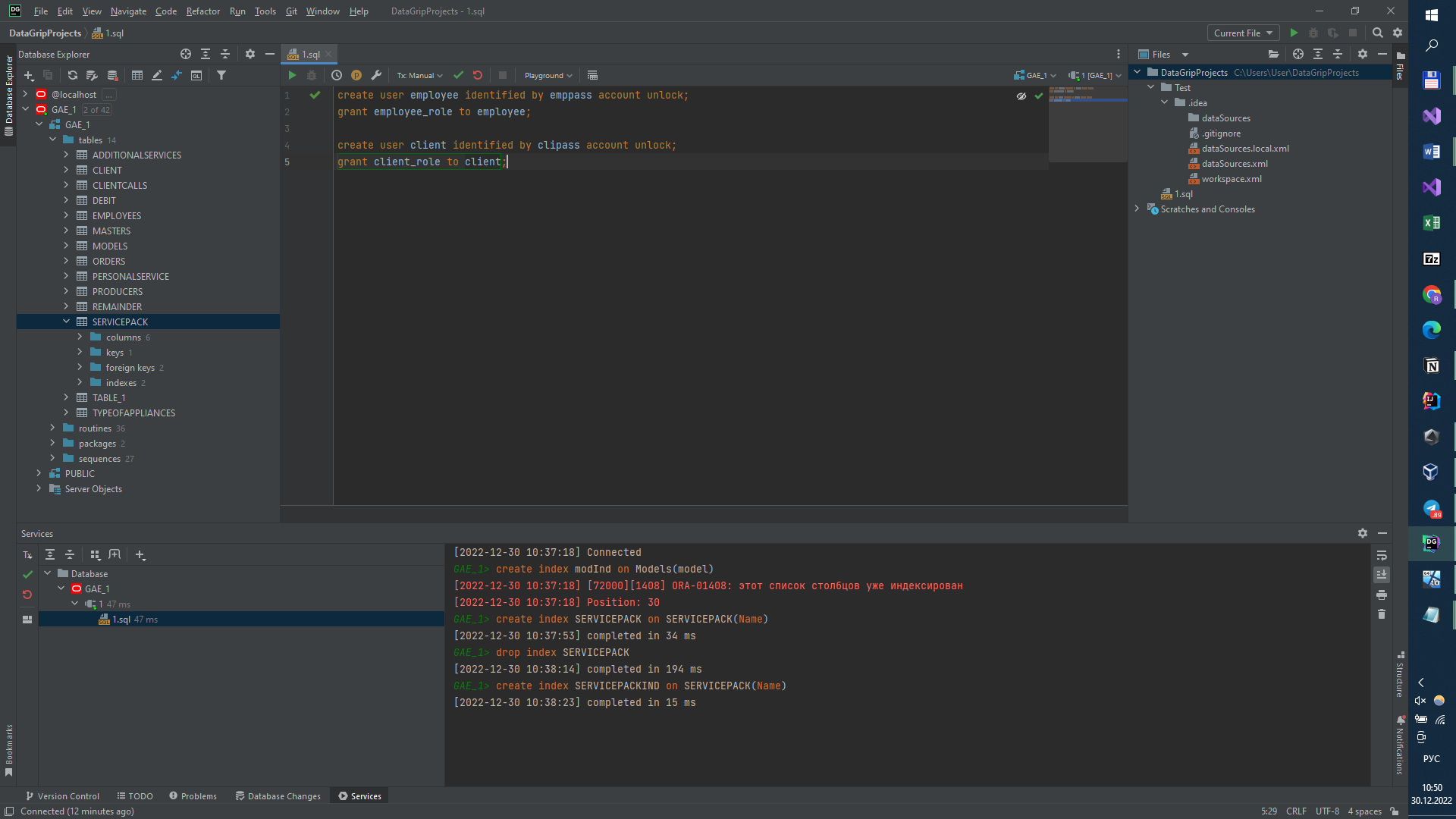


Рисунок 2.2 – Создание пользователей client и employee

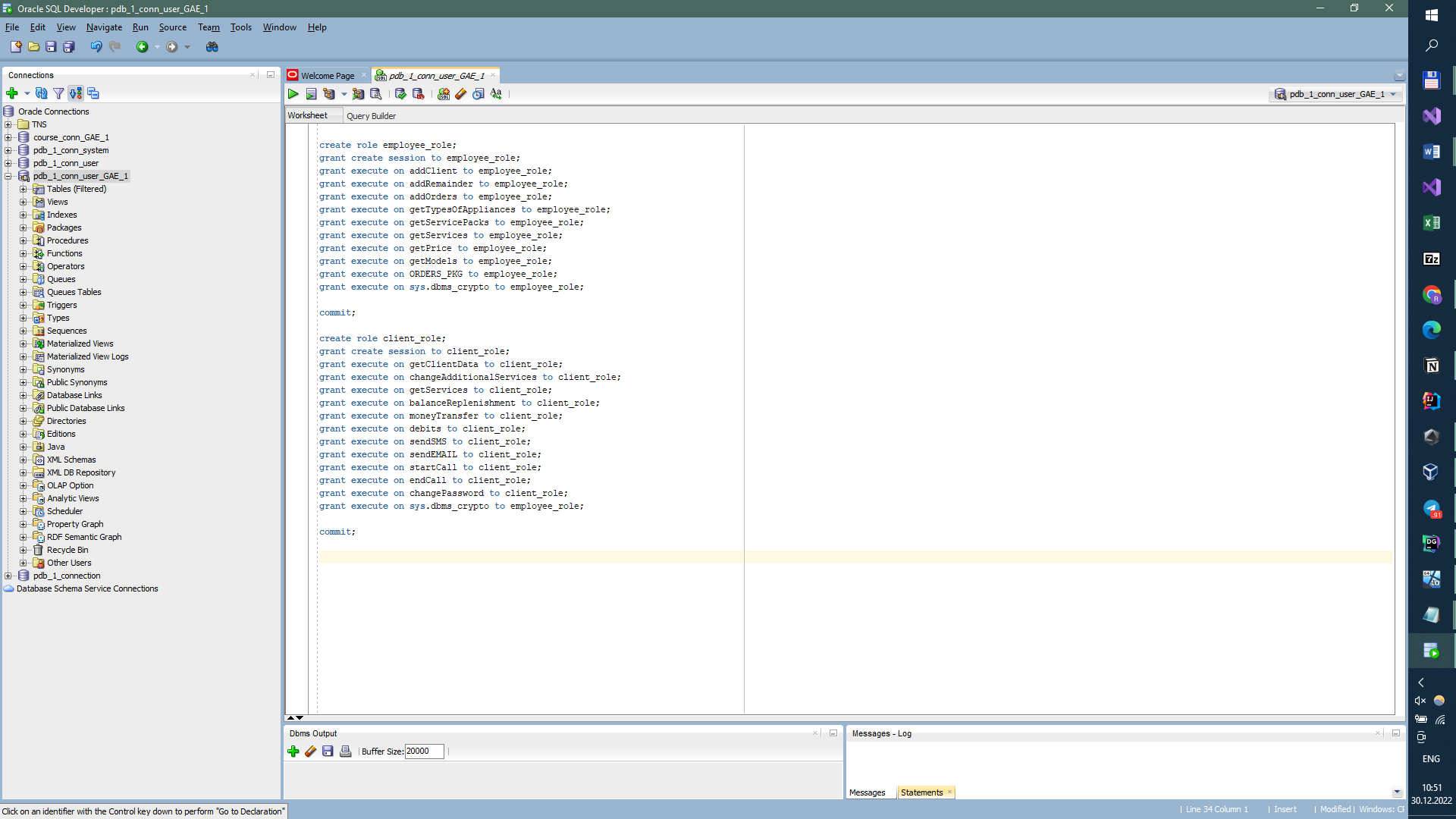


Рисунок 2.3 – Создание ролей client\_role и employee

# Технология резервного копирования и восстановления БД

Задача полного восстановления — восстановить базу данных целиком. В период восстановления база данных находится вне сети. Прежде чем какая-либо часть базы данных перейдет в режим «в сети», все данные восстанавливаются до точки согласования, в которой все части базы данных находятся в одном и том же моменте времени и в которой нет незафиксированных транзакций.

При работе в режиме модели полного восстановления после восстановления резервных копий данных необходимо восстановить все последующие резервные копии журнала транзакций, а затем саму базу данных. Базу данных можно восстановить до определенной точки восстановления в одной из этих резервных копий журналов. Этой точкой восстановления может быть заданная дата и время, помеченная транзакция или регистрационный номер транзакции в журнале (LSN).

При восстановлении базы данных, в особенности при использовании модели полного восстановления или модели восстановления с неполным протоколированием, следует использовать одну последовательность восстановления. Последовательность восстановления состоит из одной или нескольких операций восстановления, которые выполняют перемещение данных в одном или нескольких этапах восстановления.

Пример полного восстановления представлен на рисунке 3.1.

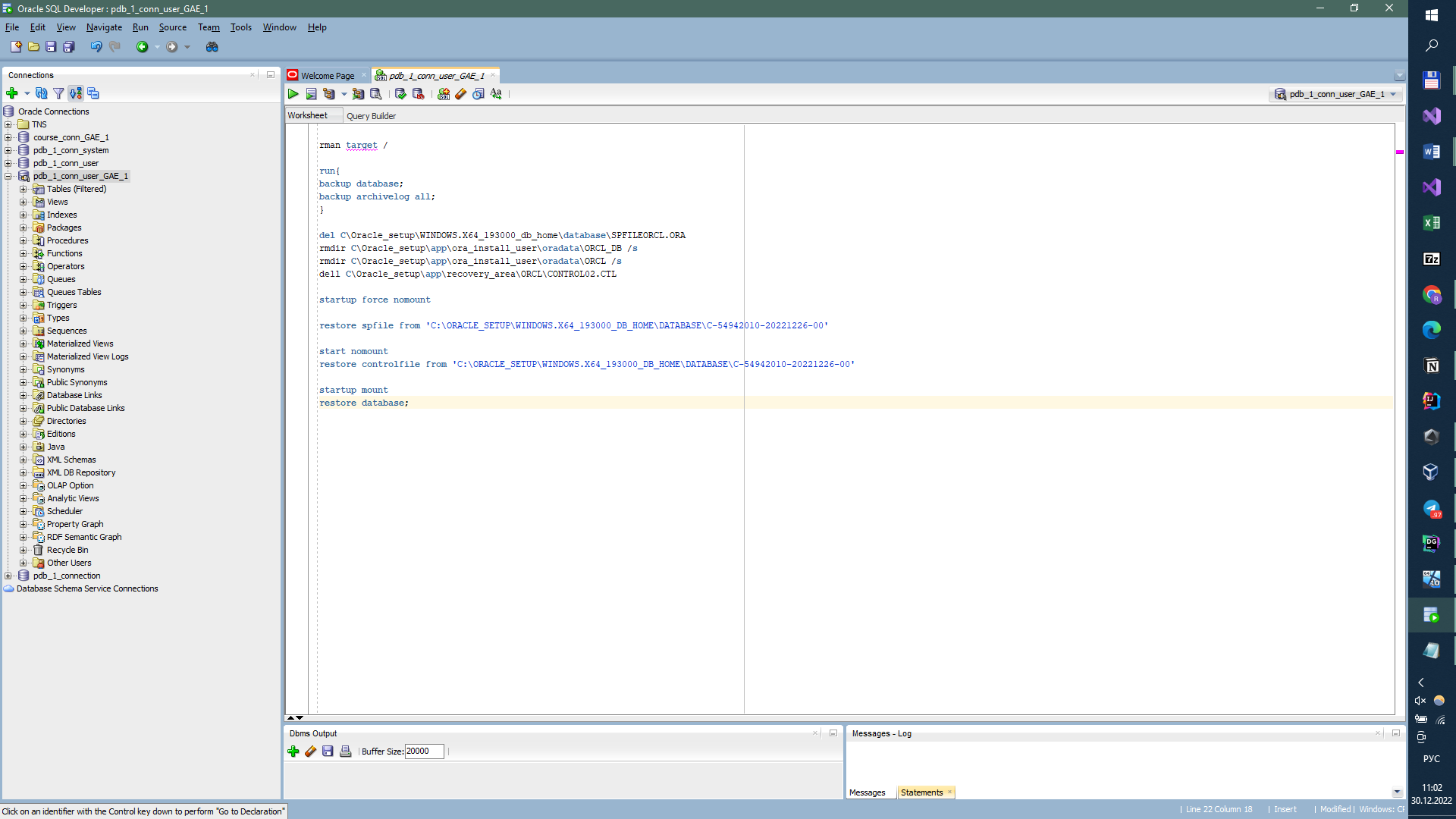


Рисунок 3.1 – Пример скрипта полного восстановления

# Импорт и экспорт данных

Для импорта таблицы Services в формат xml была разработана процедура IMPORT\_SERVICES \_FROM\_XML в составе пакета XML\_PACKAGE. Процедура IMPORT\_ SERVICES \_FROM\_XML представлена ниже на рисунке 4.1. Отрывок из xml представлен в приложении Г.

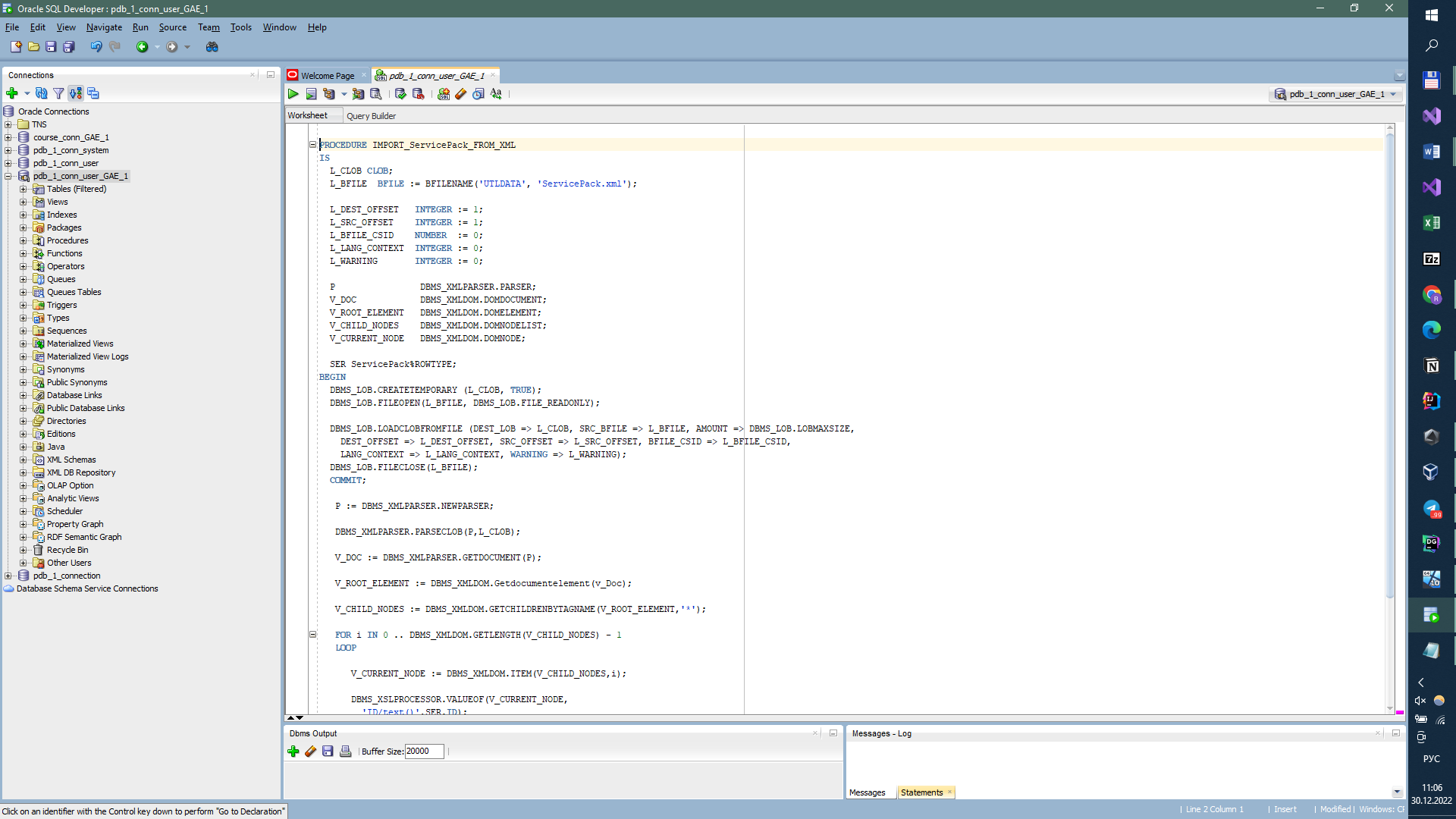


Рисунок 4.1 – Пример создания процедуры IMPORT\_SERVICES \_FROM\_XML

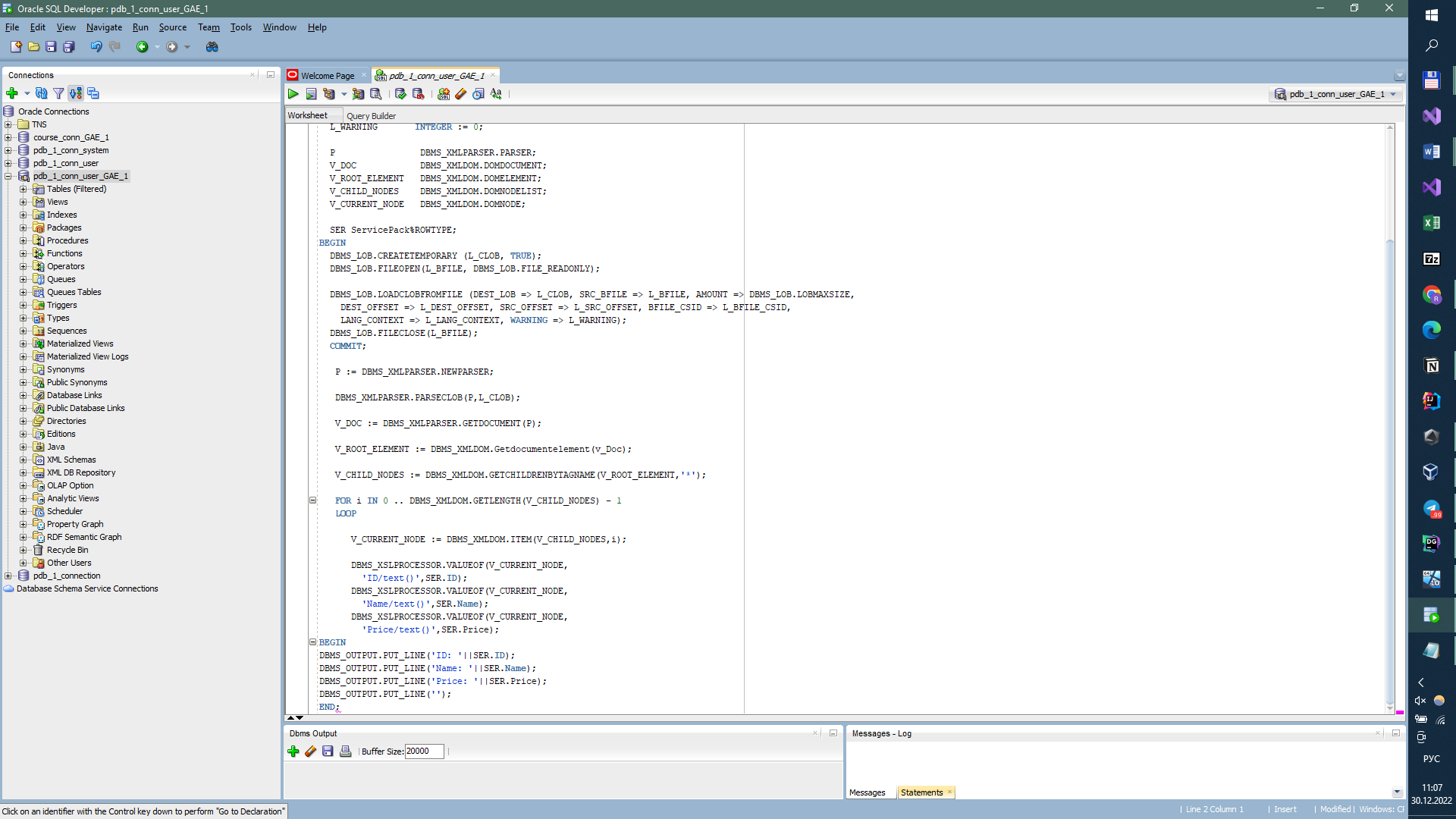


Рисунок 4.1 Продолжение – Пример создания процедуры IMPORT\_SERVICES \_FROM\_XML

Для экспорта данных в таблицу Services, в формат xml, была разработана процедура EXPORT\_SERVICES\_TO\_XML для добавления данных о услугах, зарегистрированных в приложении.

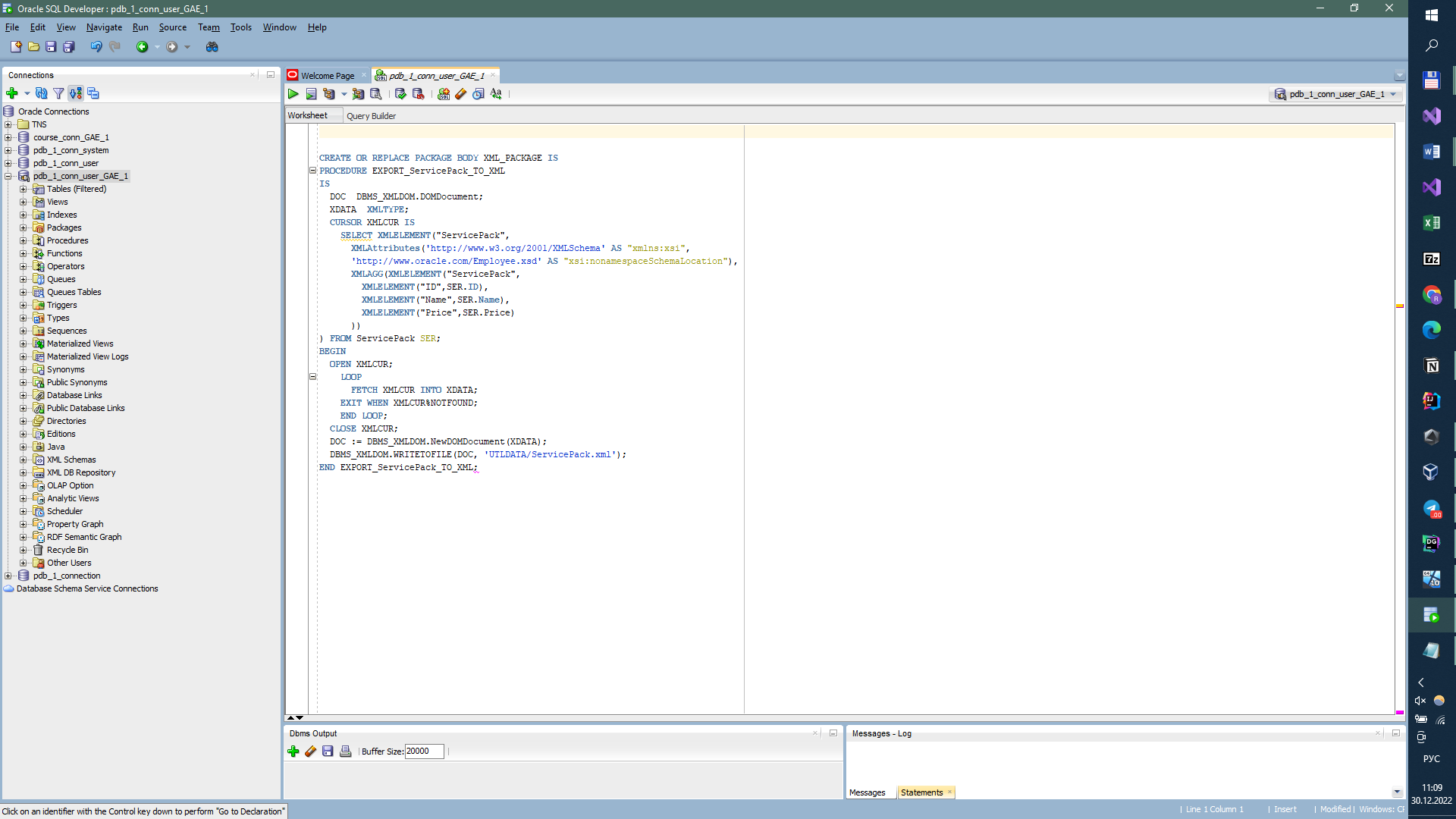


Рисунок 4.2 – Пример создания процедуры EXPORT\_SERVICES\_TO\_XML

# Тестирование производительности

Для обеспечения тестирования базы данных был проведен запрос для определения стоимости выполнения запроса. Наиболее часто используемой является процедура getServices, позволяющая вывести все услуги.

План выполнения и время выполнения процедуры запроса показаны на рисунках 5.1 и 5.2 соответственно.

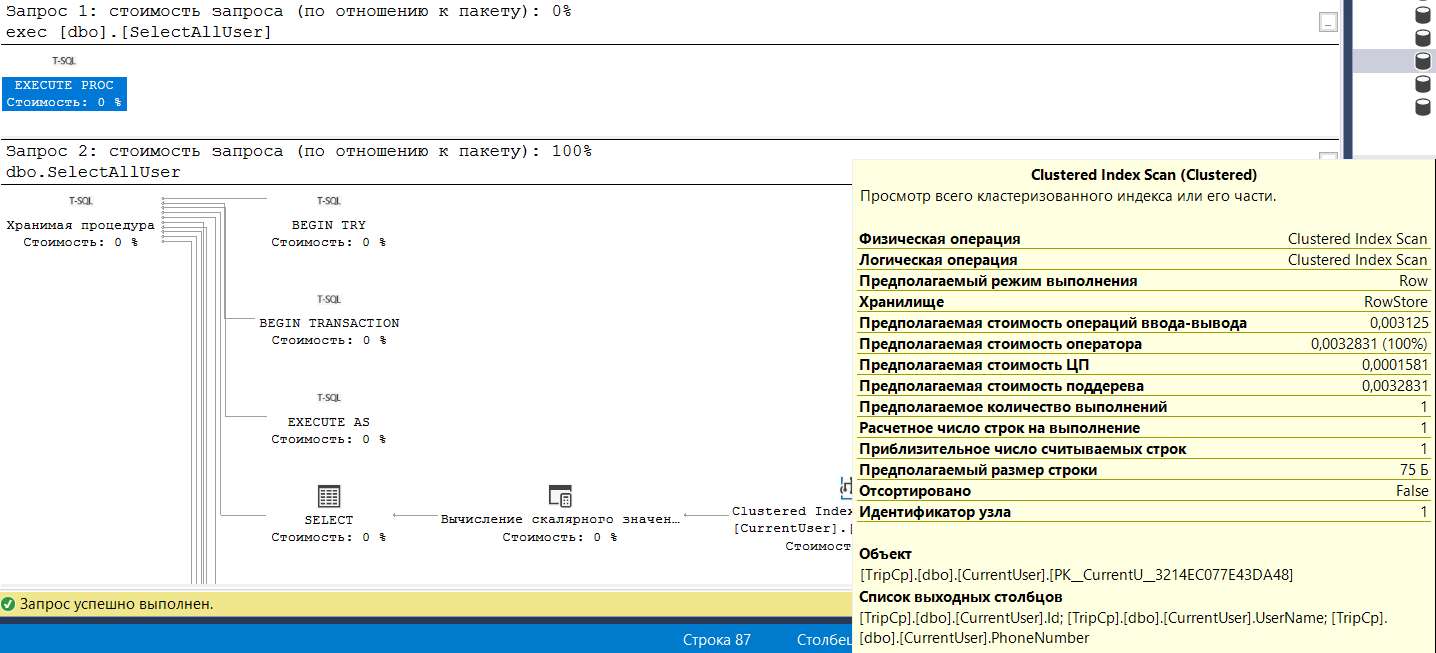


Рисунок 5.1 – План выполнения запроса

Для 1 строки результаты по предполагаемой стоимость операции ввода-вывода запроса – 0,0031, а предполагаемая стоимость ЦП запроса – 0,0032.

Анализируя результаты тестирования, можно прийти к выводу, что приложение будет работать корректно для различного объема данных.

1. **Pipelined-функции**

Для обеспечения более быстрого вывода заказов, который представляют набор разнообразных данных из разных таблиц, было решено воспользоваться технологией Pipelined в составе пакета ORDERS\_PKG, которая позволяет выводить данные построчно, что даёт значительное ускорение при больших объёмах данных.

|  |
| --- |
| create or replace PACKAGE ORDERS\_PKG AS  CURSOR cur\_ord  is  select Orders.id, Clients.PHONENUMBER,  CLIENTS.LOGIN, Orders.OrderRegDate,  MODELS.MODEL, Producers.Producer, Services.NAME,  ORDERS.OrderPrice, Orders.OrderState, ORDERS.ORDERCOMPLDATE, EMPLOYEES.ID as "EID"  from ORDERS join SERVICES on Orders.SERVICEID=Services.ID  join Models on Orders.MODELID = Models.ID  join CLIENTS on ORDERS.CLIENTID = CLIENTS.ID  join EMPLOYEES on ORDERS.EMPLOYEEID = EMPLOYEES.ID  join PRODUCERS on MODELS.PRODUCERID = PRODUCERS.ID  where Orders.EMPLOYEEID is not null and Orders.ORDERSTATE=0 and EMPLOYEES.POST='Работник';  CURSOR cur\_ord2  is  select Orders.id, Clients.PHONENUMBER,  CLIENTS.LOGIN, Orders.OrderRegDate,  MODELS.MODEL, Producers.Producer, Services.NAME,  ORDERS.OrderPrice, Orders.OrderState, ORDERS.ORDERCOMPLDATE, EMPLOYEES.ID as "EID"  from ORDERS join SERVICES on Orders.SERVICEID=Services.ID  join Models on Orders.MODELID = Models.ID  join CLIENTS on ORDERS.CLIENTID = CLIENTS.ID  join EMPLOYEES on ORDERS.EMPLOYEEID = EMPLOYEES.ID  join PRODUCERS on MODELS.PRODUCERID = PRODUCERS.ID  where Orders.EMPLOYEEID is not null and Orders.ORDERSTATE=1 and EMPLOYEES.POST='Работник';  TYPE OrderType IS RECORD(or\_id Orders.id%TYPE,  or\_num CLIENTS.PHONENUMBER%TYPE,  cl\_log CLIENTS.LOGIN%TYPE,  or\_date Orders.OrderRegDate%TYPE,  or\_model MODELS.MODEL%TYPE,  m\_prod PRODUCERS.Producer%TYPE,  serv SERVICES.NAME%TYPE,  or\_price ORDERS.OrderPrice%TYPE,  or\_state Orders.OrderState%TYPE,  or\_enddate Orders.ORDERCOMPLDATE%TYPE,  empl\_id EMPLOYEES.ID%TYPE);  TYPE TableOrderType IS TABLE OF OrderType;  function GET\_ACCEPTED\_ORDERS  return TableOrderType PIPELINED;  function GET\_READY\_ORDERS  return TableOrderType PIPELINED;  no\_masters exception;  not\_ready exception;  END ORDERS\_PKG;  create or replace PACKAGE BODY ORDERS\_PKG AS  function GET\_ACCEPTED\_ORDERS  return TableOrderType pipelined is  out\_rec OrderType;  roww cur\_ord%rowtype;  c number;  Begin  select count(\*) into c from MASTERS where MASTERS.EMPLOYEESID > 0;  if c < 1 then raise no\_masters;  end if;  Open cur\_ord;  loop  fetch cur\_ord into roww;  exit when cur\_ord%notfound;  PIPE ROW(roww);  End loop;  Close cur\_ord;  exception  when no\_masters then  raise\_application\_error(-20031, 'No masters / Мастеров не добавлено.');  end GET\_ACCEPTED\_ORDERS;  function GET\_READY\_ORDERS  return TableOrderType pipelined is  out\_rec OrderType;  roww cur\_ord2%rowtype;  Begin  Open cur\_ord2;  loop  fetch cur\_ord2 into roww;  if roww.ORDERSTATE=0 then raise not\_ready;  end if;  exit when cur\_ord2%notfound;  PIPE ROW(roww);  End loop;  Close cur\_ord2;  exception  when not\_ready then  raise\_application\_error(-20032, 'Order not ready / Заказ не готов.');  end GET\_READY\_ORDERS;  END ORDERS\_PKG;  commit;  declare  BEGIN  select \* from table(ORDERS\_PKG.GET\_accepted\_Orders);  select \* from table(ORDERS\_PKG.GET\_READY\_ORDERS);  end; |

Листинг 5.1 – План выполнения запроса

# Заключение

В рамках работы над проектом был проведен обзор аналогичных решений и программных продуктов, выбрана платформа для разработки и система управления базами данных, спроектирована архитектура и структура базы данных, проведено тестирование производительности запросов к таблицам базы данных при различном объеме данных.

Одной из главных задач данной курсовой работы - освоение технологии шифрование и маскирование данных. Данная технология и ее возможности были успешно изучены и реализованы в процессе разработки базы данных.

Данная база данных соответствует представленной задаче и отвечает всем необходимым требованиям:

* Распределением фонда заявок по категориям технических средств
* Управлением данными о персонале
* Обеспечением услугами в зависимости от типа техники
* Обработка заявок в зависимости от статуса их выполнения
* Анализ работы предприятия (анализ работы мастера по общей прибыли и по количеству обработанных заявок, анализ работы отдела, анализ конкретной услуги)

Разработанная база банных содержит в себе 10 связанных таблиц. Разработка базы данных проводилась в рамках Oracle Database 19c – система управления реляционными базами данных. Данная СУБД используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия.

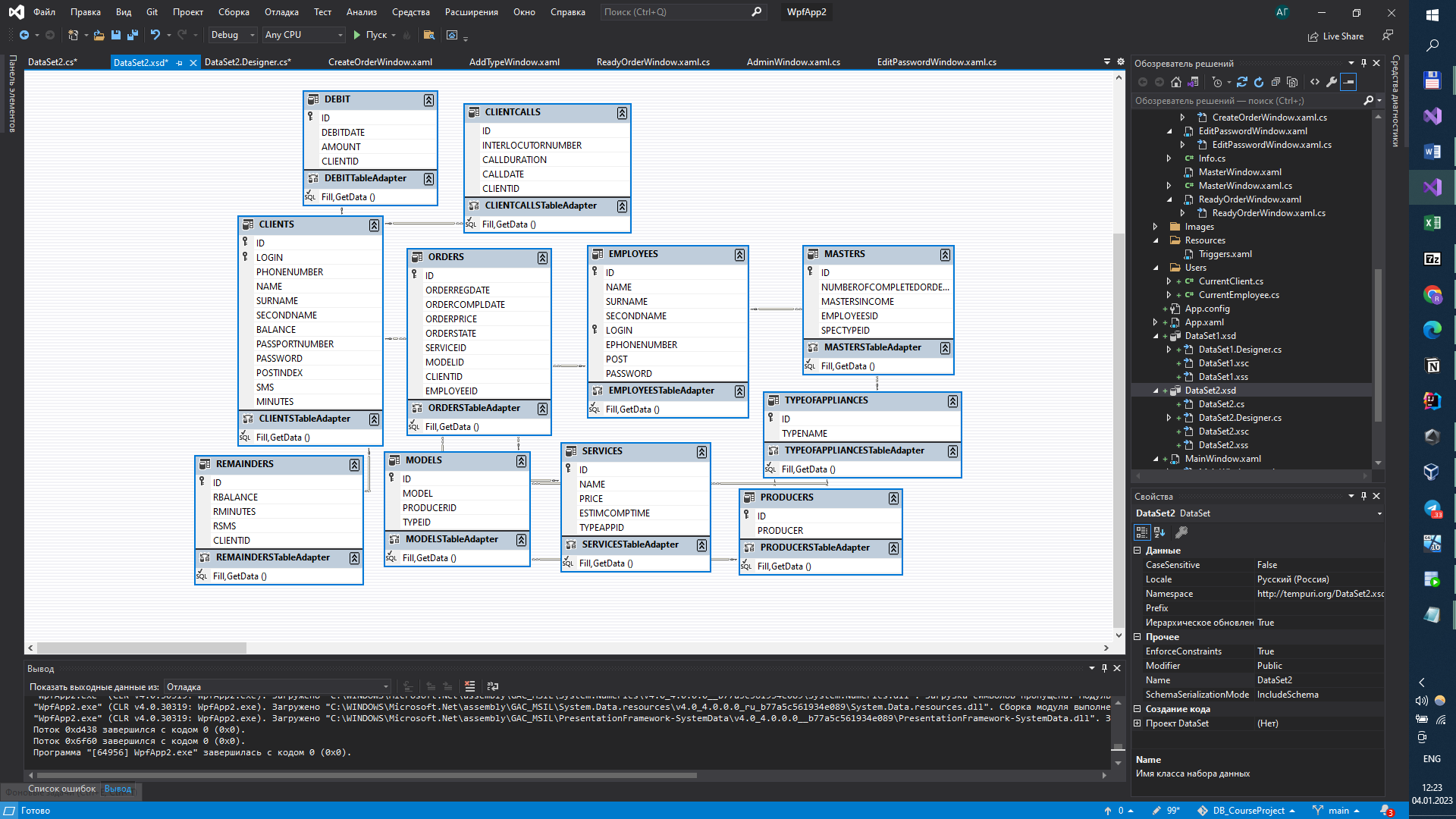
База данных законченной, хотя и возможна доработка и расширение функционала.

# Список использованных источников

1. Выполнение полного восстановления базы данных [Электронный ресурс] / Microsoft. – Режим доступа: https://learn.microsoft.com/ru-ru/sql/relational-databases/backup-restore/complete-database-restores-full-recovery-model?view=sql-server-ver16
2. Backup and Recovery User's Guide [Электронный ресурс] / Microsoft. – Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/19/bradv/database-backup-and-recovery-users-guide.pdf
3. Техническая документация по Oracle Database 19c / Oracle. – Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/

# Приложение А

Диаграмма базы данных



# Приложение Б

Процедуры авторизации и регистрации пользователей

|  |
| --- |
| create or replace procedure findUser(in\_login in Employees.Login%TYPE, in\_password in Employees.Password%TYPE, user\_cur out sys\_refcursor)  is  invalid\_user exception;  check\_count number;  encode\_pass raw(2000);  begin  encode\_pass := COURSE\_CRYPT.ENCRYPTION\_S(in\_password);  select count(\*) into check\_count from Employees where Login=in\_login and Password=encode\_pass;  if check\_count!=0 then user\_cur := get\_user\_cursor(in\_login, encode\_pass);  else raise invalid\_user;  end if;  dbms\_output.enable();  dbms\_sql.return\_result(user\_cur);  exception  when invalid\_user then  raise\_application\_error(-20000,'Проверьте введенные данные');  end findUser; |
| create or replace procedure findClient(in\_login in CLIENTS.LOGIN%TYPE, in\_password in Clients.Password%TYPE, user\_cur out sys\_refcursor)  is  invalid\_user exception;  check\_log number;  check\_phone number;  phn CLIENTS.Phonenumber%type;  encode\_pass nvarchar2(2000);  begin  encode\_pass := COURSE\_CRYPT.ENCRYPTION\_S(in\_password);  phn := in\_login;  select count(\*) into check\_log from Clients where CLIENTS.LOGIN=in\_login and Clients.Password=encode\_pass;  select count(\*) into check\_phone from Clients where CLIENTS.PHONENUMBER=in\_login and Clients.Password=encode\_pass;  if check\_log>0 then user\_cur := get\_client\_cursor(phn, in\_password);  elsif check\_phone>0 then user\_cur := get\_client\_cursor2(in\_login, in\_password);  else raise invalid\_user;  end if;  dbms\_output.enable();  dbms\_sql.return\_result(user\_cur);  exception  when invalid\_user then  raise\_application\_error(-20007,'Проверьте введенные данные');  end findClient; |

|  |
| --- |
| **create or replace procedure addClient(in\_name in Client.Name%TYPE, in\_surname in Client.Surname%TYPE, in\_secondname in Client.SecondName%TYPE,**  **in\_passportnumber in Client.PassportNumber%TYPE, in\_issuedby in Client.IssuedBy%TYPE, in\_town in Client.Town%TYPE, in\_street in Client.Street%TYPE,**  **in\_buildingnumber in Client.BuildingNumber%TYPE)**  **is**  **begin**  **insert into Client(Name, Surname, SecondName, PassportNumber, IssuedBy, Town, Street, BuildingNumber)**  **values (in\_name, in\_surname, in\_secondname, in\_passportnumber, in\_issuedby, in\_town, in\_street, in\_buildingnumber);**  **commit;**  **end;** |

|  |
| --- |
| **CREATE or replace PROCEDURE addEmployee(in\_name in Employees.Name%TYPE, in\_surname in Employees.Surname%TYPE,**  **in\_secondname in Employees.SecondName%TYPE, in\_login in Employees.Login%TYPE, in\_ephonenumber EMPLOYEES.EPHONENUMBER%Type,**  **in\_password in Employees.Password%TYPE, in\_Post in Employees.Post%TYPE)**  **IS**  **user\_exists number;**  **user\_p\_exists number;**  **curr\_user\_exists exception;**  **curr\_phone\_exists exception;**  **encode\_pass raw(2000);**  **invaliddata exception;**  **begin**  **if in\_name is null or in\_surname is null or in\_secondname is null or in\_login is null**  **or in\_password is null or in\_Post is null or in\_ephonenumber is null**  **then raise invaliddata;**  **end if;**  **encode\_pass := COURSE\_CRYPT.ENCRYPTION\_S(in\_password);**  **SELECT COUNT(\*) into user\_exists from Employees where Employees.Login = in\_login;**  **SELECT COUNT(\*) into user\_p\_exists from Employees where Employees.EPHONENUMBER = in\_ephonenumber;**  **if (user\_exists != 0 ) then raise curr\_user\_exists;**  **elsif (user\_p\_exists != 0) then raise curr\_phone\_exists;**  **else insert into Employees (Name, Surname, SecondName, Login, Password, Post,EPHONENUMBER)**  **values (in\_name, in\_surname, in\_secondname, in\_login,encode\_pass, in\_Post, in\_ephonenumber);**  **end if;**  **exception**  **when curr\_user\_exists then**  **raise\_application\_error(-20001, 'Пользователь с таким логином или номером телефона существует');**  **when curr\_phone\_exists then**  **raise\_application\_error(-20041, 'Пользователь с таким номером телефона существует');**  **when invaliddata then**  **raise\_application\_error(-20029, 'Проверьте введенные данные');**  **commit;**  **end addEmployee;** |

|  |
| --- |
| **create or replace procedure registrationClient(in\_number in CLIENTS.PHONENUMBER%TYPE,**  **in\_password in Clients.Password%TYPE, in\_login CLIENTS.Login%type)**  **IS**  **encode\_password varchar2(2000);**  **log\_exists number;**  **data\_contains number;**  **data\_exists exception;**  **login\_exists exception;**  **begin**  **encode\_password := COURSE\_CRYPT.ENCRYPTION\_S(in\_password);**  **select Count(\*) into data\_contains from CLIENTS**  **where Clients.PHONENUMBER = in\_number;**  **if data\_contains>0 then raise data\_exists;**  **end if;**  **select Count(\*) into log\_exists from CLIENTS**  **where Clients.Login = in\_login;**  **if log\_exists>0 then raise login\_exists;**  **end if;**  **if (log\_exists=0 and data\_contains=0) then insert into Clients(CLIENTS.Name, Surname, LOGIN, CLIENTS.PASSWORD, PHONENUMBER, BALANCE)**  **values ('User', 'User', in\_login , encode\_password, in\_number, 0);**  **end if;**  **exception**  **when data\_exists then**  **raise\_application\_error(-20006, 'Номер телефона уже привязан к учетной записи');**  **when login\_exists then**  **raise\_application\_error(-20042, 'Логин уже существует');**  **commit;**  **end registrationClient;**  **commit;** |

Процедуры для вывода данных

|  |
| --- |
| **create or replace procedure getClientData(in\_id in Orders.Id%TYPE, curs out sys\_refcursor)**  **is**  **begin**  **open curs for select Orders.PHONENUMBER,Remainder.Balance, Remainder.RMinutes, Remainder.RSMS,**  **Remainder.REMAIL from Orders join Remainder on Orders.RemainderId=Remainder.Id where Orders.Id=in\_id;**  **end;** |

|  |
| --- |
| **create or replace PROCEDURE getServices(service out sys\_refcursor)**  **IS**  **begin**  **open service for select Name from ServicePack;**  **dbms\_output.enable();**  **dbms\_sql.return\_result(service);**  **end getServices;**  **CREATE or replace PROCEDURE getServicePacks(servpacks out sys\_refcursor)**  **IS**  **begin**  **open servpacks for select Name from ServicePack;**  **end getServicePacks;**  **commit;** |

|  |
| --- |
| **create or replace procedure getEmployees(curs out sys\_refcursor)**  **is**  **begin**  **open curs for select Surname, Name, SecondName, Post from Employees;**  **end;** |

|  |
| --- |
| create or replace procedure getPrice(in\_servicePack in ServicePack.Name%TYPE,in\_type in TYPEOFAPPLIANCES.Name%TYPE,servpacks out sys\_refcursor)  is  begin  open servpacks for select Price from SERVICEPACK join TYPEOFAPPLIANCES T on SERVICEPACK.TYPEAPPID = T.ID  where SERVICEPACK.NAME = in\_servicePack and TYPEAPPID = in\_type;  end; |

|  |
| --- |
| create or replace PROCEDURE getTypesOfAppliances(types out sys\_refcursor)  IS  begin  open types for select TYPEOFAPPLIANCES.NAME from TYPEOFAPPLIANCES;  dbms\_output.enable();  dbms\_sql.return\_result(types);  end getTypesOfAppliances; |

|  |
| --- |
| create procedure getEmployeesBySurname(in\_empsurname in EMPLOYEES.SURNAME%TYPE,curs out sys\_refcursor)  is  begin  open curs for select EMPLOYEES.LOGIN from Employees  where EMPLOYEES.SURNAME=in\_empsurname and EMPLOYEES.POST='Работник';  end; |

|  |
| --- |
| create procedure getEmployeesByLogin(in\_emplogin in EMPLOYEES.LOGIN%TYPE,curs out sys\_refcursor)  is  begin  open curs for select EMPLOYEES.NAME,EMPLOYEES.SECONDNAME from Employees  where EMPLOYEES.LOGIN=in\_emplogin and EMPLOYEES.POST='Работник';  end; |

|  |
| --- |
| create procedure getEmployeesWorkers(curs out sys\_refcursor)  is  begin  open curs for select Surname, EMPLOYEES.NAME, SecondName, Post  from Employees where POST='Работник';  end; |

Процедуры для добавления заказа

|  |
| --- |
| create or replace PROCEDURE addOrders(in\_login in CLIENTS.LOGIN%TYPE,  in\_services in Services.Name%TYPE, in\_emplid in Orders.EmployeeId%TYPE,  in\_modelname in Models.Model%TYPE, in\_spackinfo in SERVICEPACKINFO.ID%type)  IS  tariff\_id number;  create or replace PROCEDURE addOrders(in\_login in CLIENTS.LOGIN%TYPE,  in\_services in Services.Name%TYPE, in\_emplid in Orders.EmployeeId%TYPE,  in\_modelname in Models.Model%TYPE, in\_spackinfo in SERVICEPACKINFO.ID%type)  IS  tariff\_id number;  n number;  spackinfo\_id number;  client\_id number;  master\_exists number;  not\_type number;  type\_id number;  mod\_id number;  order\_Price number;  order\_Price2 number;  number\_valid exception;  invaliddata exception;  employee\_not\_master exception;  is\_not\_master\_type exception;  begin  if in\_login is null or in\_services is null or in\_emplid is null  or in\_modelname is null or in\_spackinfo is null then raise invaliddata;  end if;  select count(\*) into master\_exists from EMPLOYEES  join MASTERS on EMPLOYEES.ID = MASTERS.EMPLOYEESID  where MASTERS.EMPLOYEESID = in\_emplid;  if master\_exists = 0 then raise employee\_not\_master;  end if;  select count(\*) into not\_type from EMPLOYEES  join MASTERS on EMPLOYEES.ID = MASTERS.EMPLOYEESID  join MODELS on MASTERS.SPECTYPEID = MODELS.TYPEID  where MODELS.MODEL = in\_modelName and EMPLOYEES.ID = in\_emplid;  if not\_type = 0 then raise is\_not\_master\_type;  end if;  select SERVICES.Id into tariff\_id from SERVICES inner join TYPEOFAPPLIANCES T  on T.ID = SERVICES.TYPEAPPID inner join MODELS M on T.ID = M.TYPEID  where SERVICES.Name=in\_services and M.MODEL =in\_modelName;  select CLIENTS.id into client\_id from CLIENTS  where CLIENTS.LOGIN = in\_login;  select SERVICES.Price into order\_Price from SERVICES where SERVICES.ID=tariff\_id;  select SERVICEPACKINFO.SPPrice into order\_Price2 from SERVICEPACKINFO  join ORDERS on SERVICEPACKINFO.ID = ORDERS.SERVICEPACKINFOID;  select Models.Id into mod\_id from Models where Models.Model=in\_modelName;  select TYPEOFAPPLIANCES.ID into type\_id from TYPEOFAPPLIANCES  join MODELS on TYPEOFAPPLIANCES.ID = MODELS.TYPEID  where MODELS.id = mod\_id;  select (MAX(ID)+1) into spackinfo\_id from SERVICEPACKINFO;  insert into SERVICEPACKINFO(id,TYPEAPPID,SPPRICE)  values (spackinfo\_id,type\_id,0);  insert into Orders(ORDERSTATE, ORDERREGDATE ,ORDERCOMPLDATE, ORDERPRICE,  SERVICEID,MODELID,EMPLOYEEID,ClientId, SERVICEPACKINFOID)  values(0, CURRENT\_TIMESTAMP, (CURRENT\_TIMESTAMP+12), order\_Price,  tariff\_id, mod\_id, in\_emplid,client\_id, spackinfo\_id);  commit;  exception  when employee\_not\_master then  raise\_application\_error(-20003, 'Текущий работник не является мастером');  when is\_not\_master\_type then  raise\_application\_error(-20004, 'Мастер существует, но не работает с данным типом приборов');  when invaliddata then  raise\_application\_error(-20030, 'Проверьте введенные данные');  end addOrders; |

Процедуры для создания

|  |
| --- |
| create procedure addMaster(in\_name in EMPLOYEES.Name%TYPE, in\_surname in EMPLOYEES.Surname%TYPE,  in\_secondname in EMPLOYEES.SecondName%TYPE, in\_empllogin EMPLOYEES.LOGIN%type,  in\_typename in TYPEOFAPPLIANCES.NAME%TYPE)  is  invaliddata exception;  curr\_master\_exists exception;  empl\_id number;  type\_id number;  begin  if in\_name is null or in\_surname is null or in\_secondname is null  or in\_empllogin is null or in\_typename is null  then raise invaliddata;  end if;  select TYPEOFAPPLIANCES.ID into type\_id from TYPEOFAPPLIANCES  where TYPEOFAPPLIANCES.NAME = in\_typename;  select EMPLOYEES.ID into empl\_id from EMPLOYEES  where EMPLOYEES.NAME = in\_name and EMPLOYEES.SURNAME = in\_surname  and EMPLOYEES.SECONDNAME = in\_secondname and EMPLOYEES.LOGIN = in\_empllogin;  insert into Masters(Masters.NUMBEROFCOMPLETEDORDERS, Masters.INCOME, Masters.EMPLOYEESID,Masters.SPECTYPEID ,Masters.TYPENAME)  values (0, 0, empl\_id, type\_id, in\_typename);  exception  when curr\_master\_exists then  raise\_application\_error(-20001, 'Такой мастер уже существует');  when invaliddata then  raise\_application\_error(-20029, 'Проверьте введенные данные');  commit;  end; |

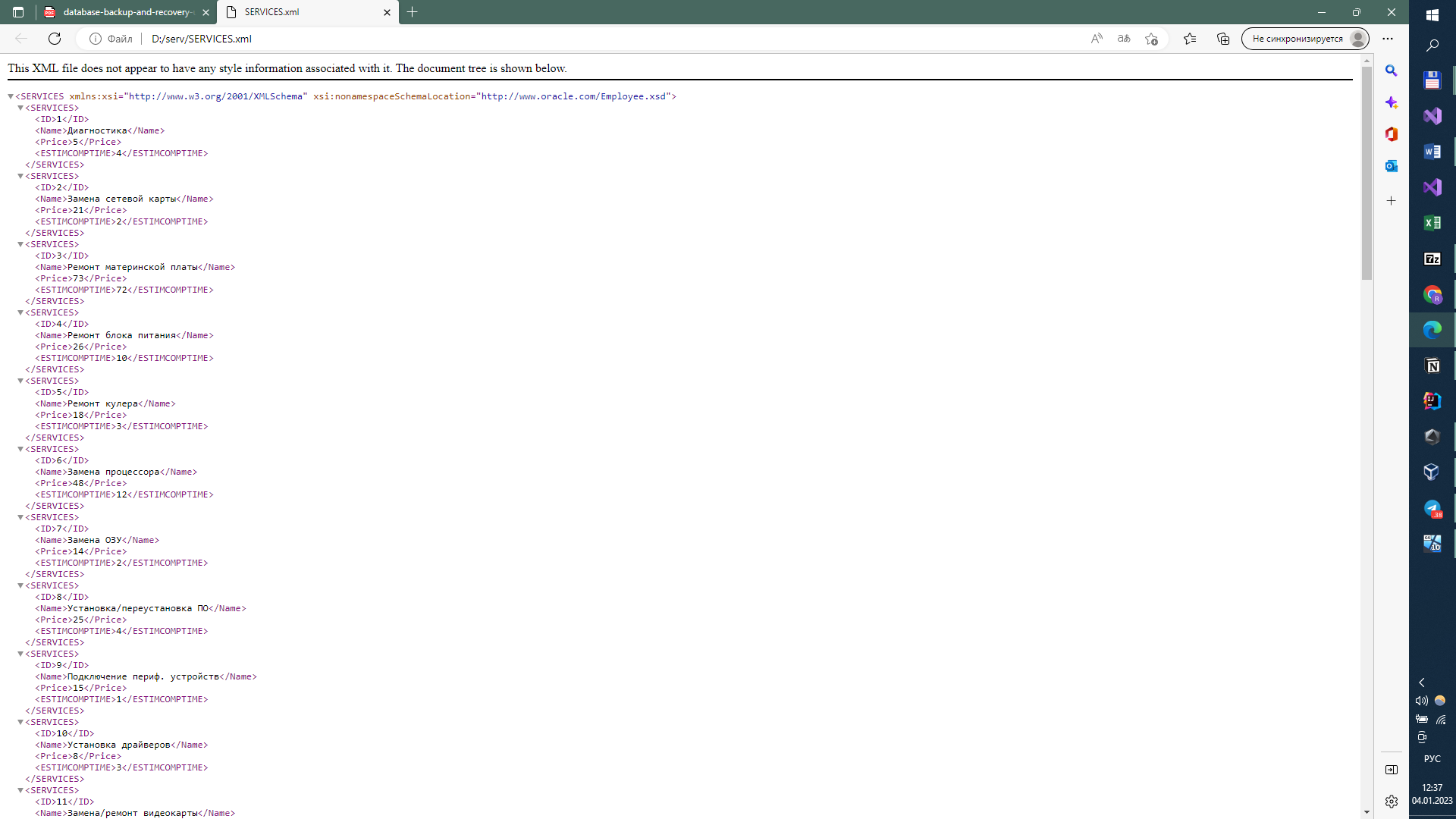
|  |
| --- |
| create PROCEDURE addService(in\_name in SERVICES.Name%TYPE, in\_price in SERVICES.Price%TYPE,  in\_typename TYPEOFAPPLIANCES.TYPENAME%TYPE, in\_time SERVICES.EstimCompTime%type)  IS  service\_exists number;  curr\_service\_exists exception;  invaliddata exception;  in\_type SERVICES.TypeAPPID%TYPE;  begin  if in\_name is null or in\_price is null or in\_typename is null or in\_time is null  then raise invaliddata;  end if;  SELECT COUNT(\*) into service\_exists from SERVICES where SERVICES.Name = in\_name;  if service\_exists != 0 then raise curr\_service\_exists;  end if;  select TYPEOFAPPLIANCES.ID into in\_type from TYPEOFAPPLIANCES  where TYPEOFAPPLIANCES.TYPENAME=in\_typename;  insert into Services(Name, Price,ESTIMCOMPTIME,TYPEAPPID)  values (in\_name,in\_price,in\_time, in\_type);  exception  when curr\_service\_exists then  raise\_application\_error(-20002, 'Такой набор услуг уже существует');  when invaliddata then  raise\_application\_error(-20028, 'Проверьте введенные данные');  commit;  end addService; |

# Приложение В

Процедура экспорта данных в XML

|  |
| --- |
| CREATE OR REPLACE PACKAGE BODY XML\_PACKAGE IS  PROCEDURE EXPORT\_Services\_TO\_XML  IS  DOC DBMS\_XMLDOM.DOMDocument;  XDATA XMLTYPE;  CURSOR XMLCUR IS  SELECT XMLELEMENT("SERVICES",  XMLAttributes('http://www.w3.org/2001/XMLSchema' AS "xmlns:xsi",  'http://www.oracle.com/Employee.xsd' AS "xsi:nonamespaceSchemaLocation"),  XMLAGG(XMLELEMENT("SERVICES",  XMLELEMENT("ID",SER.ID),  XMLELEMENT("Name",SER.Name),  XMLELEMENT("Price",SER.Price),  XMLELEMENT("ESTIMCOMPTIME",SER.ESTIMCOMPTIME),  ))  ) FROM SERVICES SER;  BEGIN  OPEN XMLCUR;  LOOP  FETCH XMLCUR INTO XDATA;  EXIT WHEN XMLCUR%NOTFOUND;  END LOOP;  CLOSE XMLCUR;  DOC := DBMS\_XMLDOM.NewDOMDocument(XDATA);  DBMS\_XMLDOM.WRITETOFILE(DOC, 'UTLDATA/SERVICES.xml');  END EXPORT\_SERVICES\_TO\_XML; |

# Приложение Г



# Приложение Д

Пакет шифрования/дешифрования данных

|  |
| --- |
| create or replace PACKAGE COURSE\_CRYPT AS  key\_bytes\_raw raw(32) := '88CBDA29C1EC8E65F4A865EC3FCC5C9F71AE6B7E5407EA445E87B17F2221AE14';  function ENCRYPTION\_S(input\_str varchar2)  return raw;  function DECRYPTION(input\_dec raw)  return varchar2;  END COURSE\_CRYPT;  create or replace PACKAGE BODY COURSE\_CRYPT AS  function DECRYPTION(input\_dec raw)  return varchar2 is  output\_string VARCHAR2 (2000);  decrypted\_raw RAW (2000); -- stores decrypted binary text  encryption\_type PLS\_INTEGER := -- total encryption type  DBMS\_CRYPTO.ENCRYPT\_AES256 + DBMS\_CRYPTO.CHAIN\_CBC + DBMS\_CRYPTO.PAD\_PKCS5;  begin  decrypted\_raw := DBMS\_CRYPTO.DECRYPT  (  input\_dec,  encryption\_type,  key\_bytes\_raw  );  output\_string := UTL\_I18N.RAW\_TO\_CHAR (decrypted\_raw, 'AL32UTF8');  return output\_string;  end DECRYPTION;  function ENCRYPTION\_S(input\_str varchar2)  return raw is  encoded\_string raw(2000);  encryption\_type PLS\_INTEGER := -- total encryption type  DBMS\_CRYPTO.ENCRYPT\_AES256 + DBMS\_CRYPTO.CHAIN\_CBC + DBMS\_CRYPTO.PAD\_PKCS5;  begin  encoded\_string:= DBMS\_CRYPTO.ENCRYPT  (  UTL\_I18N.STRING\_TO\_RAW (input\_str, 'AL32UTF8'),  encryption\_type,  key\_bytes\_raw  );  return encoded\_string;  end ENCRYPTION\_S;  END COURSE\_CRYPT; |