МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 05 01 Информационные системы и технологии

Специализация Базы данных

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«База данных для программного средства «Ресторан» с применением технологии шифрования и маскирования данных»

Выполнил студент Другаков Денис Дмитриевич

(Ф.И.О.)

Руководитель проекта асс. Н.И. Уласевич

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

Консультант: асс. Н.И. Уласевич.

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

Нормоконтролер: асс. Н.И. Уласевич.

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

Курсовой проект защищен с оценкой

Содержание

[Введение 5](#_Toc134993486)

[1 Постановка задачи 6](#_Toc134993487)

[1.1 Описание функционала и объектов базы данных 6](#_Toc134993488)

[1.2 Описание используемых технологий 7](#_Toc134993489)

[2 Проектирование базы данных 8](#_Toc134993490)

[2.1 Описание структуры базы данных 8](#_Toc134993491)

[3 Разработка объектов базы данных 10](#_Toc134993492)

[3.1 Таблицы 10](#_Toc134993493)

[3.2 Процедуры 12](#_Toc134993494)

[3.3 Пакеты 14](#_Toc134993495)

[3.4 Тригеры 15](#_Toc134993496)

[3.5 Директории 15](#_Toc134993497)

[3.6 Роли 15](#_Toc134993498)

[3.7 Пользователи 16](#_Toc134993499)

[3.8 Последовательности 16](#_Toc134993500)

[4 Описание процедур импорта и экспорта 17](#_Toc134993501)

[4.1 Процедура импорта данных из XML-файла 17](#_Toc134993502)

[4.2 Процедура экспорта данных в формате XML 17](#_Toc134993503)

[5 Тестирование производительности 19](#_Toc134993504)

[5.1 Тестирование производительности базы данных 19](#_Toc134993505)

[6 Описание технологии и ее применения в базе данных 20](#_Toc134993506)

[6.1 Шифрование данных 20](#_Toc134993507)

[6.2 Маскирование данных 21](#_Toc134993508)

[7 Краткое описание приложения для демонстрации 23](#_Toc134993509)

[7.1 Пример работы приложения от лица менеджера 23](#_Toc134993510)

[8 Руководство пользователя 25](#_Toc134993511)

[8.1 Руководство использования приложения 25](#_Toc134993512)

[Заключение 26](#_Toc134993513)

[Список используемых источников 27](#_Toc134993514)

[Приложение А Листинг создания таблиц 28](#_Toc134993515)

[Приложение Б Листинг пакета для шифрования данных 30](#_Toc134993516)

[Приложение В Листинг создания триггеров 31](#_Toc134993517)

[Приложение Г Листинг процедуры для общего импорта информации 33](#_Toc134993518)

[Приложение Д Листинг процедуры для заполнения таблиц 34](#_Toc134993519)

[Приложение Е Листинг процедуры извлечение информации из файла 36](#_Toc134993520)

[Приложение Ж Листинг процедуры получения информации из файла 37](#_Toc134993521)

[Приложение З Листинг политик маскирования 38](#_Toc134993522)

# Введение

Современное хранение информации в основном осуществляется в цифровом формате, и для эффективного управления и хранения такой информации широко используются различные базы данных. В частности, базы данных успешно применяются в приложениях для хранения и управления данными.

В рамках данного учебно-проектного задания предлагается разработать базу данных для ресторана с использованием системы управления базами данных Oracle и технологии шифрования и маскирования данных. Для того чтобы база данных была эффективной и соответствовала целям проекта, ее данные должны отвечать следующим требованиям:

* Быть полезными, т.е. сокращать информационную энтропию системы;
* Обладать полнотой, достаточной для осуществления качественного управления;
* Обеспечивать точность, достоверность и непротиворечивость данных, т.е. в базе данных не должно быть заведомо ошибочных, противоречивых или устаревших данных;
* Быть актуальными, т.е. регулярно обновляться и поддерживаться в актуальном состоянии.
* Безопасность: база данных должна обладать высоким уровнем защиты от несанкционированного доступа и быть защищена от вирусов, злоумышленников и других угроз безопасности;
* Масштабируемость: база данных должна быть способна масштабироваться и расширяться в соответствии с ростом бизнеса и увеличением объема данных. Это позволит обеспечить стабильность и надежность работы базы данных в будущем.

В соответствии с заданием курсового проекта для проектирования базы данных ресторана используется система управления базами данных Oracle Database и технология шифрования и маскирования данных.

# Постановка задачи

* 1. **Описание функционала и объектов базы данных**

Целью данного курсового проекта является проектирование базы данных для программного средства ресторана с применением технологии шифрования и маскирования данных. В качестве модели данных следует использовать реляционную модель. Проектирование необходимо произвести таким образом, чтобы конечные данные соответствовали общим требованиям к информации в базе данных

Функционал должен позволять:

* Регистрация нового пользователя;
* Просмотр меню;
* Просмотр меню по типу блюда;
* Бронирование стола;
* Оставление отзыва;
* Просмотр отзывов;
* Просмотр меню;
* Просмотр меню по типу блюда;
* Бронирование стола;
* Просмотр списка заказов;
* Обновление статуса стола;
* Расчет общей стоимости заказа;
* Добавление новых блюд;
* Добавление новых типов блюд;
* Добавление новых столов;
* Добавление новых сотрудников;
* Просмотр списка пользователей;
* Удаление пользователей;
* Удаление блюд;
* Удаление типов блюд;
* Удаление заказов;
* Удаление отзывов;
* Удаление смен;
* Удаление столов;
* Обновление информации о пользователях;
* Обновление информации о блюдах;
* Обновление информации о типах блюд;
* Обновление информации о заказах;
* Обновление информации об отзывах;
* Обновление информации о сменах;
* Обновление информации о столах.

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

* роли;
* пользователи;
* таблицы;
* представления;
* табличные пространства;
* индексы;
* хранимые процедуры;
* триггеры.
  1. **Описание используемых технологий**

Существует огромное количество систем управления базами данных (СУБД), которые могут быть использованы для организации работы с данными в компьютерных системах. Некоторые из наиболее популярных СУБД включают в себя Oracle Database, MS SQL Server, NoSQL, MySQL, PostgreSQL и другие.

В данной работе было принято решение использовать Oracle Database в качестве СУБД, так как она предоставляет широкий спектр возможностей для оперирования объектами базы данных, включая поддержку стандарта SQL, хранение и управление табличными данными, поддержку транзакций и многое другое.

Одним из ключевых аспектов при работе с базами данных является безопасность. В процессе выбора технологии СУБД была уделена особое внимание технологиям шифрования и маскирования данных. Шифрование может быть использовано для защиты данных от несанкционированного доступа путем кодирования данных с использованием алгоритмов шифрования. Маскирование, с другой стороны, может быть использовано для замены конфиденциальных данных (например, имена и адреса) на фиктивные данные, которые сохраняют формат и структуру оригинальных данных, но не раскрывают личную информацию.

# Проектирование базы данных

## Описание структуры базы данных

Для реализации поставленной задачи было создано табличное пространство PLATINUM\_KITCHEN. Диаграмма структуры полученной базы данных, разработанной в СУБД «Oracle database», представлена на рисунке 2.1. Для базы данных было разработано 9 таблиц, которые связаны друг с другом внешними ключами.

Таблица EMPLOYEES хранит данные о сотрудниках.

Таблица CUSTOMERS содержит информацию о клиентах.

Таблица TABLES хранит информацию о столах.

Таблица MENUTYPE хранит информацию о типах блюд для более удобного поиска блюд.

Таблица MENU хранит информацию о блюдах, доступных в ресторане.

Таблица ORDERS хранит информацию о заказах, сделанных клиентами.

Таблица ORDERITEMS содержит информацию о конкретных блюдах, которые были заказаны в рамках заказа.

Таблица REVIEWS хранит отзывы, оставленные клиентами о заказе и блюдах.

Таблица SHIFTS хранит данные о рабочих сменах сотрудников, включая поваров и официантов.

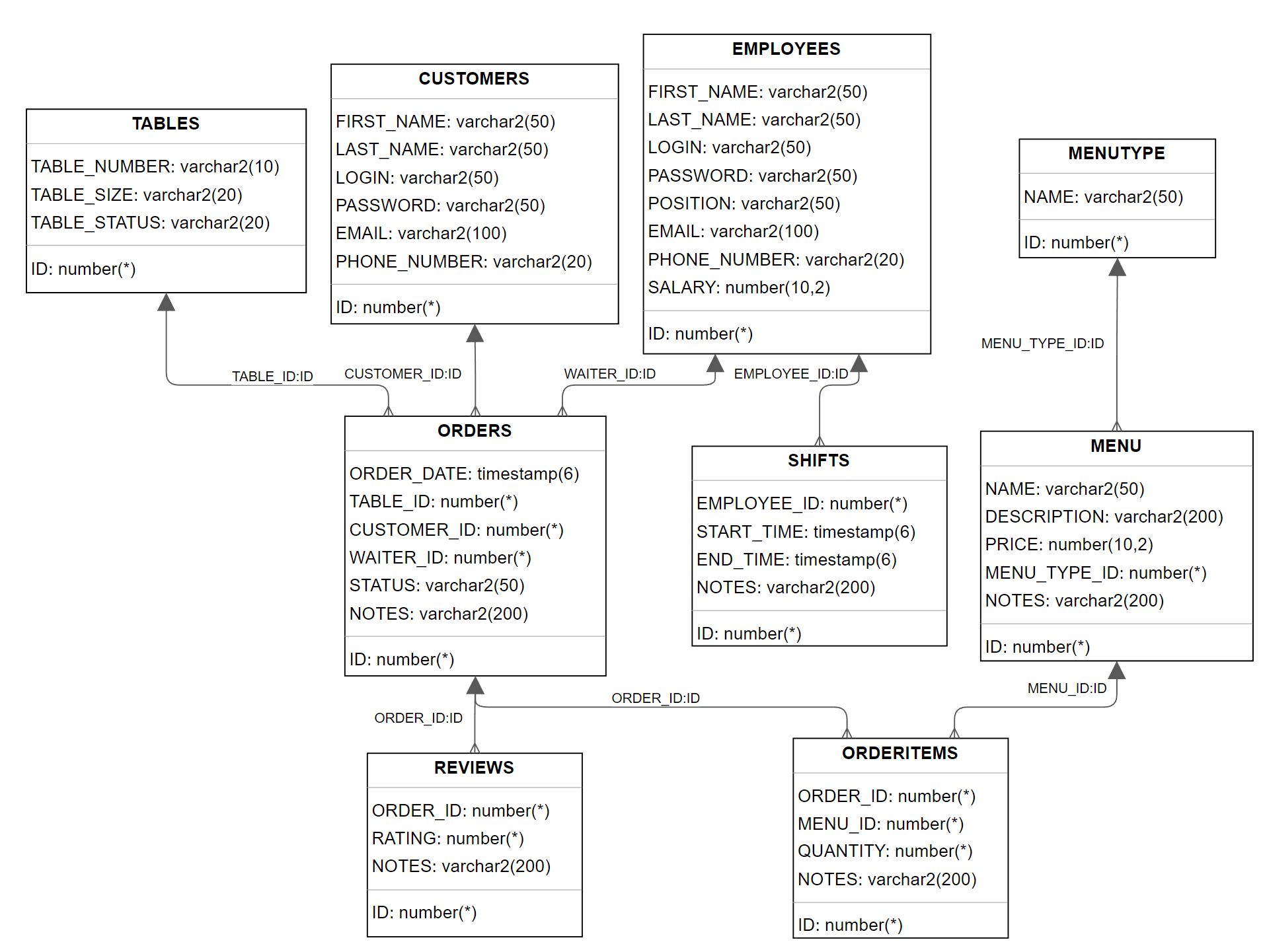


Рисунок 2.1 – Диаграмма структуры базы данных

Все таблицы связаны между собой с помощью внешних ключей, что позволяет эффективно и быстро обрабатывать данные и выполнять запросы к базе данных.

Для более наглядного представления возможностей каждой роли в системе, необходимо предоставить диаграмму вариантов использования продукта, показа она на рисунке 2.2. На данной диаграмме должны быть отображены все возможности, которые доступны конкретному пользователю с учетом его роли, будь это обычный пользователь или администратор.

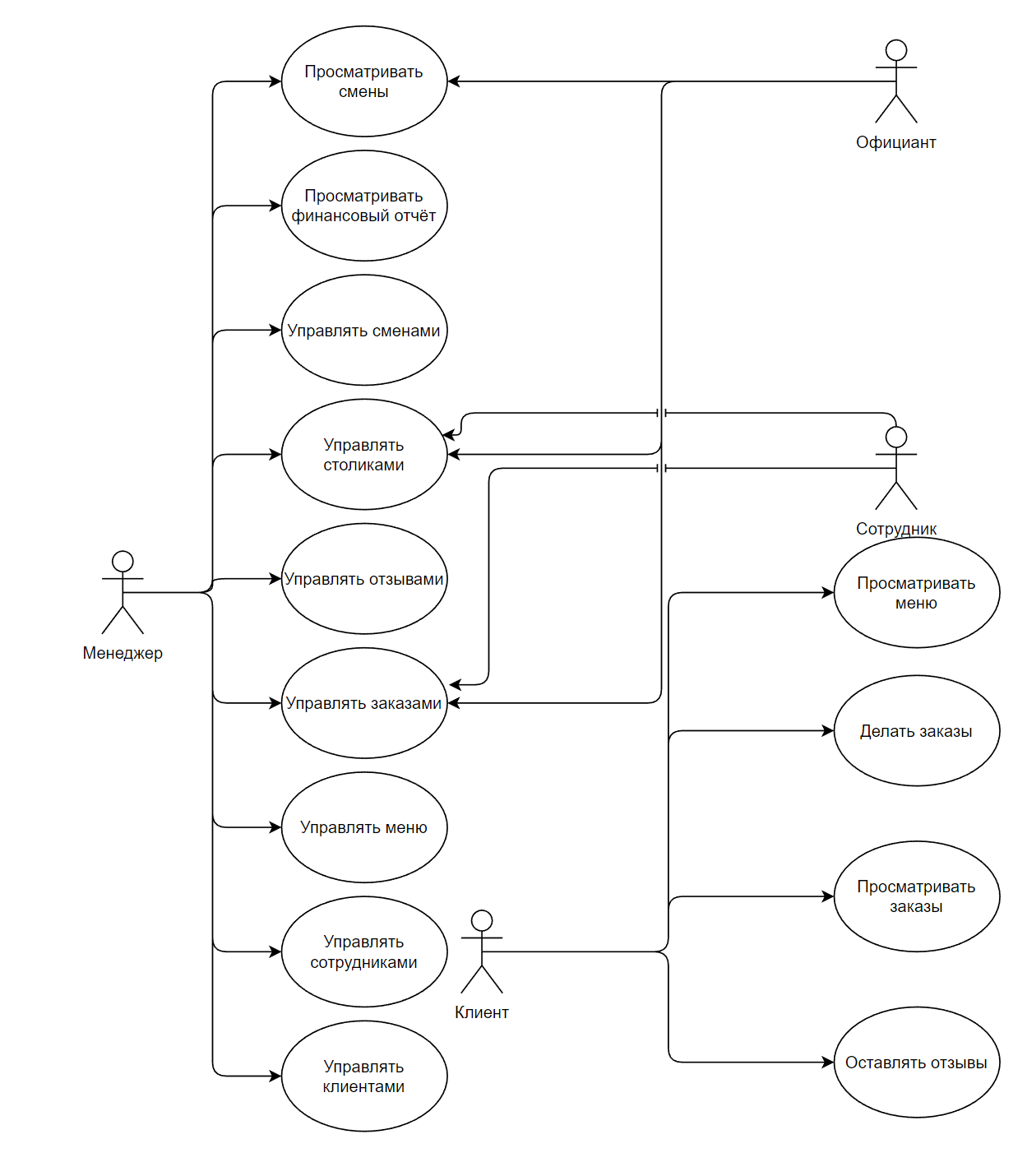


Рисунок 2.1 – Диаграмма вариантов использования

Такая диаграмма позволит легче воспринимать возможности каждой роли и понимать, какие возможности расширяются при переходе на более высокую роль в системе.

# Разработка объектов базы данных

## Таблицы

Таблицы являются неотъемлемой частью любой реляционной базы данных. Краткая характеристика каждой из таблиц была предоставлена в разделе 2, а код их создания можно увидеть в Приложении А. Ниже мы рассмотрим каждую таблицу подробнее.

Таблица EMPLOYEES состоит из девяти столбцов и представлена на таблице 3.1:

Таблица 3.1 «EMPLOYEES»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | NUMBER | Хранит идентификатор сотрудника |
| FIRST\_NAME | VARCHAR2 | Хранит имя сотрудника |
| LAST\_NAME | VARCHAR2 | Хранит фамилию сотрудника |
| LOGIN | VARCHAR2 | Хранит логин сотрудника |
| PASSWORD | VARCHAR2 | Хранит пароль сотрудника |
| POSITION | VARCHAR2 | Хранит должность сотрудника |
| EMAIL | VARCHAR2 | Хранит адрес электронной почты сотрудника |
| PHONE\_NUMBER | VARCHAR2 | Хранит номер телефона сотрудника |
| SALARY | NUMBER | Хранит зарплату сотрудника |

Таблица CUSTOMERS состоит из семи столбцов и показана на таблице 3.2:

Таблица 3.2 «CUSTOMERS»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | NUMBER | Хранит идентификатор клиента |
| FIRST\_NAME | VARCHAR2 | Хранит имя клиента |
| LAST\_NAME | VARCHAR2 | Хранит фамилию клиента |
| LOGIN | VARCHAR2 | Хранит логин клиента |
| PASSWORD | VARCHAR2 | Хранит пароль клиента |
| EMAIL | VARCHAR2 | Хранит адрес электронной почты клиента |
| PHONE\_NUMBER | VARCHAR2 | Хранит номер телефона клиента |

Таблица TABLES состоит из четырех столбцов и представлена на таблице 3.3:

Таблица 3.3 «TABLES»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | NUMBER | хранит идентификатор стола |
| TABLE\_NUMBER | VARCHAR2 | хранит номер стола |
| TABLE\_SIZE | VARCHAR2 | хранит размер стола |
| TABLE\_STATUS | VARCHAR2 | хранит статус стола |

Таблица MENUTYPE состоит из двух столбцов и представлена на таблице 3.4:

Таблица 3.4 «MENUTYPE»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | NUMBER | хранит идентификатор типа меню |
| NAME | VARCHAR2 | хранит название типа меню |

Таблица MENU состоит из шести столбцов и представлена на таблице 3.5:

Таблица 3.5 «MENU»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | NUMBER | Хранит идентификатор меню |
| NAME | VARCHAR2 | Хранит название блюда в меню |
| DESCRIPTION | VARCHAR2 | Хранит описание блюда в меню |
| PRICE | NUMBER | Хранит стоимость блюда в меню |
| MENU\_TYPE\_ID | VARCHAR2 | Хранит тип меню, внешний ключ к таблице MENUTYPE |
| NOTES | VARCHAR2 | Хранит примечания к блюду в меню |

Таблица ORDERS состоит из семи столбцов и представлена на таблице 3.6:

3.6 «ORDERS»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | NUMBER | Хранит идентификатор заказа |
| ORDER\_DATE | TIMESTAMP | Хранит дату заказа |
| TABLE\_ID | NUMBER | Хранит идентификатор стола, внешний ключ к таблице TABLES |
| CUSTOMER\_ID | NUMBER | Хранит идентификатор клиента, внешний ключ к таблице CUSTOMERS |
| WAITER\_ID | NUMBER | Хранит идентификатор официанта, внешний ключ к таблице EMPLOYEES |
| STATUS | VARCHAR2 | Хранит статус заказа |
| NOTES | VARCHAR2 | Хранит примечания к заказу |

Таблица ORDERITEMS состоит из пяти столбцов и представлена на таблице 3.7:

Таблица 3.7 «ORDERITEMS»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | NUMBER | Хранит идентификатор элемента заказа |
| ORDER\_ID | NUMBER | Хранит идентификатор заказа, внешний ключ к таблице ORDERS |
| MENU\_ID | NUMBER | Хранит идентификатор блюда, внешний ключ к таблице MENU |
| QUANTITY | NUMBER | Хранит количество заказанного блюда |
| NOTES | VARCHAR2 | Хранит примечания к элементу заказа |

Таблица REVIEWS состоит из четырёх столбцов и представлена на таблице 3.8:

Таблица 3.8 «REVIEWS»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | NUMBER | Хранит идентификатор отзыва |
| ORDER\_ID | NUMBER | Хранит идентификатор заказа, внешний ключ к таблице ORDERS |
| RATING | NUMBER | Хранит рейтинг заказа |
| NOTES | VARCHAR2 | Хранит примечания к отзыву |

Таблица SHIFTS состоит из пяти столбцов и представлена на таблице 3.9

Таблица 3.9 «SHIFTS»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | NUMBER | Хранит идентификатор смены |
| EMPLOYEE\_ID | NUMBER | Хранит идентификатор сотрудника, внешний ключ к таблице EMPLOYEES |
| START\_TIME | TIMESTAMP | Хранит время начала смены |
| END\_TIME | TIMESTAMP | Хранит время окончания смены |
| NOTES | VARCHAR2 | Хранит примечания к смене |

## Процедуры

Всего было разработано 64 процедуры для работы с данными. Краткое описание процедур описано далее.

Пример создания хранимой процедуры Calculate\_Order\_Total, которая добавляет блюда представлена в листинге 3.1

|  |
| --- |
| create PROCEDURE Calculate\_Order\_Total(p\_order\_id IN NUMBER,p\_total\_price OUT NUMBER)AS BEGIN  SELECT SUM(OrderItems.quantity \* Menu.price) INTO p\_total\_price  FROM OrderItems  JOIN Menu ON OrderItems.menu\_id = Menu.id  WHERE OrderItems.order\_id = p\_order\_id;END; |

Листинг 3.1 – Процедура Calculate\_Order\_Total

Данная процедура Calculate\_Order\_Total предназначена для вычисления итоговой суммы заказа, основываясь на информации о количестве заказанных блюд и их ценах. Входным параметром является p\_order\_id - идентификатор заказа, для которого необходимо вычислить итоговую сумму. Выходным параметром является p\_total\_price - итоговая сумма заказа, которая будет передана в вызывающий код.

Внутри процедуры осуществляется выборка из таблицы OrderItems и таблицы Menu, где связующим элементом является идентификатор заказа (p\_order\_id) и идентификатор блюда (menu\_id). Затем для каждого блюда учитывается количество (quantity) и его цена (price). Общая сумма всех заказанных блюд сохраняется в переменную p\_total\_price и передается наружу в вызывающий код.

ADD\_CUSTOMER - процедура для добавления нового клиента в базу данных.

ADD\_EMPLOYEE - процедура для добавления нового сотрудника в базу данных.

ADD\_MENU\_ITEM - процедура для добавления нового блюда в меню.

ADD\_MENU\_TYPE - процедура для добавления нового типа блюда в базу данных.

ADD\_ORDER - процедура для создания нового заказа.

ADD\_ORDER\_ITEM - процедура для добавления нового блюда в заказ.

ADD\_REVIEW - процедура для добавления нового отзыва в базу данных.

ADD\_SHIFT - процедура для добавления новой смены в базу данных.

ADD\_TABLE - процедура для добавления нового столика в базу данных.

ASSIGN\_WAITER\_TO\_ORDER - процедура для назначения официанта на заказ.

CALCULATE\_ORDER\_TOTAL - процедура для вычисления итоговой суммы заказа.

CANCEL\_RESERVATION - процедура для отмены резервации столика.

CHECK\_CUSTOMER\_LOGIN\_PASSWORD - процедура для проверки логина и пароля клиента.

CHECK\_EMPLOYEE\_LOGIN\_PASSWORD - процедура для проверки логина и пароля сотрудника.

CLEAR\_ALL\_TABLES\_AND\_SEQUENCES - процедура для очистки всех таблиц и секвенций.

DELETE\_CUSTOMER - процедура для удаления клиента из базы данных.

DELETE\_EMPLOYEE - процедура для удаления сотрудника из базы данных.

DELETE\_MENU\_ITEM - процедура для удаления блюда из меню.

DELETE\_MENU\_TYPE - процедура для удаления типа блюда из базы данных.

DELETE\_ORDER - процедура для удаления заказа из базы данных.

DELETE\_REVIEW - процедура для удаления отзыва из базы данных.

DELETE\_SHIFT - процедура для удаления смены из базы данных.

DELETE\_TABLE - процедура для удаления столика из базы данных.

EXPORT\_ALL\_TABLES - процедура для экспорта всех таблиц в xml-файл.

EXPORT\_TO\_FILE - процедура для экспорта данных из таблицы в xml-файл.

FILE\_TO\_CLOB - процедура для импорта данных из xml-файла в таблицу.

FINANCIAL\_REPORT - процедура для генерации финансового отчета.

GET\_ACTIVE\_ORDERS - процедура для получения всех активных заказов.

GET\_ALL\_MENU\_ITEMS - процедура для получения всех блюд из меню.

GET\_ALL\_ORDERS - процедура для получения всех заказов из базы данных.

GET\_ALL\_REVIEWS - процедура для получения всех отзывов из базы данных.

GET\_FREE\_TABLES - процедура для получения всех свободных столиков.

GET\_MENU\_BY\_TYPE - процедура для получения блюд из меню по типу блюда.

GET\_MENU\_ITEM\_BY\_ID - процедура для получения блюда из меню по ID.

GET\_ORDERS\_BY\_CUSTOMER\_ID - процедура для получения заказов клиента по ID.

GET\_REVIEWS\_BY\_ID - процедура для получения отзывов по ID.

GET\_SHIFT\_BY\_EMPLOYEE\_ID - процедура для получения смены сотрудника по его ID.

GET\_TABLE\_STATUS - процедура для получения статуса столика.

DELETE\_ORDER\_ITEM - процедура для удаления блюда из заказа.

GET\_TABLES\_BY\_STATUS - процедура для получения списка столов с определенным статусом.

IMPORT\_ALL\_XML\_DATA - процедура для импорта всех данных из XML-файлов в базу данных.

IMPORT\_CUSTOMERS\_XML - процедура для импорта данных клиентов из XML-файла в базу данных.

IMPORT\_EMPLOYEE\_XML - процедура для импорта данных сотрудников из XML-файла в базу данных.

IMPORT\_MENU\_TYPE\_XML - процедура для импорта данных типов блюд из XML-файла в базу данных.

IMPORT\_MENU\_XML - процедура для импорта данных блюд из XML-файла в меню.

IMPORT\_ORDER\_ITEMS\_XML - процедура для импорта данных блюд в заказе из XML-файла в базу данных.

IMPORT\_ORDERS\_XML - процедура для импорта данных заказов из XML-файла в базу данных.

IMPORT\_REVIEWS\_XML - процедура для импорта данных отзывов из XML-файла в базу данных.

IMPORT\_SHIFTS\_XML - процедура для импорта данных смен из XML-файла в базу данных.

IMPORT\_TABLES\_XML - процедура для импорта данных столов из XML-файла в базу данных.

PAY\_ORDER - процедура для оплаты заказа.

POPULATE\_ALL\_TABLES - процедура для заполнения базы данных тестовыми данными.

RESERVE\_TABLE - процедура для резервации столика.

RESET\_SEQ - процедура для сброса счетчиков последовательностей.

UPDATE\_CUSTOMER - процедура для обновления данных клиента в базе данных.

UPDATE\_EMPLOYEE - процедура для обновления данных сотрудника в базе данных.

UPDATE\_MENU\_ITEM - процедура для обновления данных блюда в меню.

UPDATE\_MENU\_TYPE - процедура для обновления данных типа блюда в базе данных.

UPDATE\_ORDER - процедура для обновления данных заказа в базе данных.

UPDATE\_ORDER\_ITEM - процедура для обновления данных о блюде в заказе.

UPDATE\_REVIEW - процедура для обновления данных об отзыве.

UPDATE\_SHIFT - процедура для обновления данных о смене.

UPDATE\_TABLE - процедура для обновления данных о столике.

UPDATE\_TABLE\_STATUS - процедура для обновления статуса столика в базе данных,

## Пакеты

Всего был создан один пакет, который содержит две функции - ENCRYPT и DECRYPT, код реализации представлен в приложении Б.

Функция ENCRYPT предназначена для шифрования переданной строки в формат RAW с помощью алгоритма AES128 в режиме цепочки CBC с дополнением PKCS5.

Функция DECRYPT, в свою очередь, предназначена для расшифровки полученного зашифрованного текста в формате RAW.

Таким образом, данный пакет предназначен для шифрования и дешифровки текстовых данных с использованием алгоритма.

## Тригеры

Всего была разработано 4 триггера для шифрования логина и пароля пользователей и сотрудников при их вставке и обновлении в таблицы "CUSTOMERS" и "EMPLOYEES". Триггеры используются для обеспечения безопасности хранения конфиденциальных данных пользователей и сотрудников в базе данных. Реализация триггеров представлена в приложение В.

## Директории

Всего была создана одна директория, показанная в листинге 3.2, для указания пути к файлам, содержащим данные импортированные из таблиц.

|  |
| --- |
| CREATE DIRECTORY DATA\_DIR AS 'C:\APP\DENIS\ADMIN\ORCL\BACKUP'; |

Листинг 3.2 – Директория указывающая на резервные файлы

Одним из преимуществ использования директории для хранения данных является то, что это позволяет значительно сократить объем кода, который необходим для импорта и экспорта данных в базе данных. Вместо того, чтобы указывать путь к файлу каждый раз при выполнении импорта или экспорта данных, вы можете просто указать директорию, которая содержит все необходимые файлы.

## Роли

Создание ролей - это важный аспект безопасности баз данных. Роли используются для управления доступом пользователей к объектам базы данных, таким как таблицы, представления, процедуры и т.д. В Oracle 12c для создания ролей используется команда CREATE ROLE.

Всего было реализовано ролей 6 ролей:

* C##CLIENT\_ROLE – для клиентов,
* C##EMPLOYEE\_ROLE – для работников,
* C##MANAGER\_ROLE – для менеджеров,
* C##WAITER\_ROLE – для официантов,
* C##PLATINUM\_KITCHEN\_MASKING – специальная роль, пользователи с этой ролью смог видеть не замаскированные данные,
* C##Platinum\_Kitchen\_Admin -для администратора;

Каждой роли были назначены определенные привилегии, которые позволяют пользователям выполнять определенные операции в базе данных. Например, роль C##CLIENT\_ROLE имеет доступ только к операциям, связанным с клиентами, таким как создание, удаление и обновление данных клиентов, создание заказов и т.д. Роль C##MANAGER\_ROLE имеет права на создание и удаление отзывов, а также просмотр всех заказов и меню. Роль C##WAITER\_ROLE имеет права на просмотр активных заказов и статуса столиков.

Кроме того, мы создали роль C##Platinum\_Kitchen\_Admin, которая имеет все привилегии, доступные другим пользователям. Эта роль может использоваться только администраторами базы данных и должна быть назначена только определенным пользователям.

Таким образом, создание ролей является важным шагом в обеспечении безопасности баз данных и помогает управлять доступом пользователей к объектам базы данных. Важно тщательно продумать и назначить правильные привилегии для каждой роли, чтобы предотвратить несанкционированный доступ к данным и защитить конфиденциальность данных.

## Пользователи

Создание пользователей является важным шагом в обеспечении безопасности и контроля доступа к базе данных. В данной базе данных созданы пять пользователей, каждый из которых имеет свои уникальные права доступа.

Каждый клиент ресторана "Platinum Kitchen" может создать свою учетную запись в базе данных, используя логин и пароль. Для этого был создан пользователь C##CLIENT\_ROLE, который будет использоваться клиентами для входа в систему и получения доступа к своим заказам и другой персональной информации.

Второй пользователь C##EMPLOYEE\_ROLE создан для сотрудников ресторана "Platinum Kitchen". Он будет использоваться для доступа к информации о меню, заказах и других операциях, связанных с работой ресторана. У этого пользователя есть доступ к различным процедурам базы данных, таким как ADD\_MENU\_ITEM, ADD\_ORDER и другим.

Третий пользователь C##MANAGER\_ROLE создан для менеджеров ресторана "Platinum Kitchen". Он имеет доступ к той же информации, что и C##EMPLOYEE\_ROLE, а также дополнительные права, позволяющие ему добавлять и удалять отзывы о ресторане.

Четвертый пользователь C##WAITER\_ROLE создан для официантов ресторана "Platinum Kitchen". Он имеет доступ к информации о заказах и статусу столиков, а также может изменять статусы заказов и отдельных блюд.

Пятый пользователь C##Platinum\_Kitchen\_Admin создан для администратора базы данных. Он имеет полный доступ ко всей базе данных.

## Последовательности

Для автоматического инкрементирования числовых идентификаторов в таблицах можно использовать последовательности. Для этого мы создадим последовательности для каждой из наших таблиц.

# Описание процедур импорта и экспорта

При всей своей отлаженности и очевидности, классический способ хранения и представления объектов развитой структуры имеет и вполне определенные недостатки и может вызывать проблемы, с которыми сталкивался любой разработчик, пытавшийся реализовать таким способом достаточно сложную систему. В некоторых ситуациях, решить эти проблемы позволяет хранение объекта в виде XML.

## Процедура импорта данных из XML-файла

Для импорта XML-данных в базу данных Oracle используется функция XMLTYPE, которая преобразует содержимое XML-файла в объект XMLTYPE. Для обработки этого объекта используется оператор EXTRACT, который извлекает значения соответствующих элементов XML и помещает их в переменные. Для каждой таблицы создается отдельная процедура, которая принимает параметром XML-файл, из которого происходит импорт данных. Так-же была написана процедура для универсального извлечения информации из файла и последующего использования. Процедура для общего импорта, извлечения информации из файла, а также один пример обработки информации были представлены в Приложение Г.

## Процедура экспорта данных в формате XML

В базе данных Oracle были разработаны две процедуры, предназначенные для экспорта данных из базы в формате XML. Первая процедура EXPORT\_TO\_FILE, представленная в листинге 4.1, принимает входные параметры запрос и имя файла, и возвращает результат выполнения запроса в виде XML-строки. Эта процедура создает файл с указанным именем в директории DATA\_DIR и записывает в него результат выполнения запроса в формате XML.

|  |
| --- |
| create PROCEDURE EXPORT\_TO\_FILE(  p\_query IN VARCHAR2,  p\_filename IN VARCHAR2  )  AS  v\_clob NCLOB;  v\_file UTL\_FILE.FILE\_TYPE;  BEGIN  SELECT DBMS\_XMLGEN.GETXML(p\_query) INTO v\_clob FROM DUAL;  v\_file := UTL\_FILE.FOPEN('DATA\_DIR', p\_filename || '.xml', 'w');  BEGIN  UTL\_FILE.PUT(v\_file, v\_clob);  EXCEPTION  WHEN UTL\_FILE.WRITE\_ERROR THEN  DBMS\_OUTPUT.PUT\_LINE('Error writing to file');  END;  UTL\_FILE.FCLOSE(v\_file);  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('Query returned no data');  WHEN UTL\_FILE.INVALID\_PATH THEN  DBMS\_OUTPUT.PUT\_LINE('Invalid path for directory');  WHEN UTL\_FILE.INVALID\_MODE THEN  DBMS\_OUTPUT.PUT\_LINE('Invalid file mode');  WHEN UTL\_FILE.INVALID\_FILEHANDLE THEN  DBMS\_OUTPUT.PUT\_LINE('Invalid file handle');  WHEN UTL\_FILE.INVALID\_OPERATION THEN  DBMS\_OUTPUT.PUT\_LINE('Invalid file operation');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Unknown error: ' || SQLERRM);  END; |

Листинг 4.1 – Процедура EXPORT\_TO\_FILE

Вторая процедура EXPORT\_ALL\_TABLES представленная в листинге 4.2, использует процедуру EXPORT\_TO\_FILE для экспорта данных всех таблиц в базе данных.

|  |
| --- |
| create PROCEDURE EXPORT\_ALL\_TABLES  AS  BEGIN  EXPORT\_TO\_FILE('SELECT ID, FIRST\_NAME, LAST\_NAME,' ||  ' MYCRYPTOPACKAGE.DECRYPT(UTL\_RAW.CAST\_TO\_RAW(lOGIN)) as lOGIN,' ||  ' MYCRYPTOPACKAGE.DECRYPT(UTL\_RAW.CAST\_TO\_RAW(PASSWORD)) as PASSWORD,' ||' EMAIL, PHONE\_NUMBER FROM CUSTOMERS', 'CUSTOMERS');  EXPORT\_TO\_FILE('SELECT ID, FIRST\_NAME, LAST\_NAME,' ||  ' MYCRYPTOPACKAGE.DECRYPT(UTL\_RAW.CAST\_TO\_RAW(lOGIN)) as lOGIN,' ||  ' MYCRYPTOPACKAGE.DECRYPT(UTL\_RAW.CAST\_TO\_RAW(PASSWORD)) as PASSWORD,' ||' POSITION, EMAIL, PHONE\_NUMBER, SALARY FROM EMPLOYEES', 'EMPLOYEES');  EXPORT\_TO\_FILE('SELECT \* FROM MENU', 'MENU');  EXPORT\_TO\_FILE('SELECT \* FROM MENUTYPE', 'MENUTYPE');  EXPORT\_TO\_FILE('SELECT \* FROM ORDERITEMS', 'ORDERITEMS');  EXPORT\_TO\_FILE('SELECT \* FROM ORDERS', 'ORDERS');  EXPORT\_TO\_FILE('SELECT \* FROM REVIEWS', 'REVIEWS');  EXPORT\_TO\_FILE('SELECT \* FROM SHIFTS', 'SHIFTS');  EXPORT\_TO\_FILE('SELECT \* FROM TABLES', 'TABLES');  END; |

Листинг 4.2 – Процедура EXPORT\_ALL\_TABLES

Для каждой таблицы вызывается процедура EXPORT\_TO\_FILE с соответствующим запросом и именем файла, чтобы экспортировать данные этой таблицы в отдельный XML-файл.Эта процедура экспортирует данные из таблиц CUSTOMERS, EMPLOYEES, MENU, MENUTYPE, ORDERITEMS, ORDERS, REVIEWS, SHIFTS и TABLES.

# Тестирование производительности

## Тестирование производительности базы данных

Для тестирования производительности базы данных, нам нужно заполнить таблицы достаточным количеством записей. Для данного случая мы заполнили каждую таблицу по 100.000 строк, скрипт заполнения таблиц приложен в приложение Д.

Далее, для улучшения производительности запросов, мы создали несколько индексов. Для таблицы ORDERS мы создали индексы на столбцы ORDER\_DATE, TABLE\_ID, CUSTOMER\_ID, WAITER\_ID и STATUS. Для таблицы ORDERITEMS мы создали индекс на столбец ORDER\_ID, а для таблицы REVIEWS - на столбец ORDER\_ID. Для таблицы SHIFTS мы создали индексы на столбцы EMPLOYEE\_ID и START\_TIME. Пример для создания индексов показан в листинге 5.1.

Теперь давайте рассмотрим производительность базы данных в контексте запросов на выборку данных.

|  |
| --- |
| CREATE INDEX idx\_orders\_order\_date ON ORDERS (ORDER\_DATE); |

Листинг 5.1 – Индекс для таблицы

При выполнении запроса показанного далее в листинге 5.1.2 на выборку данных из нескольких таблиц без использования индексов, время ожидания может значительно увеличиваться из-за необходимости выполнения большого количества операций сканирования таблиц. Так, при выполнении запроса на выборку данных из таблиц ORDERS, CUSTOMERS, TABLES, ORDERITEMS и MENU без использования индексов, время выполнения, представленное на рисунке 5.1, составило около 355 миллисекунд.



Рисунок 5.1 – Оценка запроса без использования индексов

Создание индексов на столбцах CUSTOMER\_ID, TABLE\_ID и PRICE позволило сократить время выполнения запроса на выборку данных из таблиц ORDERS, CUSTOMERS, TABLES, ORDERITEMS и MENU до 242миллисекунд, как видно на рисунке 5.2.



Рисунок 5.2 – Оценка запроса с использованием индексов

Таким образом, использование индексов является важным приемом оптимизации производительности базы данных и может сократить время выполнения запросов на выборку данных из таблиц.

# Описание технологии и ее применения в базе данных

## Шифрование данных

Для обеспечения безопасности данных в базе данных используется пакет криптографических функций DBMS\_CRYPTO, предоставляемый Oracle Database. Он позволяет использовать различные алгоритмы шифрования, хэширования и цифровой подписи.

Для защиты данных в столбцах LOGIN и PASSWORD таблиц CUSTOMERS и EMPLOYEES от несанкционированного доступа, мы добавляем триггеры, которые шифруют данные перед их добавлением в базу данных. Также мы добавляем триггеры для обновления данных в этих столбцах, чтобы новые данные также были зашифрованы перед их сохранением.

Для шифрования используется алгоритм AES128 в режиме шифрования CBC и метод заполнения PAD\_PKCS5. Для генерации ключа шифрования используется строка 'dddden3546', которая перед шифрованием объединяется со строкой 'saltys'. Таким образом, ключ шифрования формируется путем конкатенации этих двух строк. Данные необходимые для шифрования и дешифрования хранятся вне функций ENCRYPT и DECRYPT в пакете, их содержание представлено в листинге 6.1.

|  |
| --- |
| ENCRYPTION\_TYPE PLS\_INTEGER := DBMS\_CRYPTO.ENCRYPT\_AES128  + DBMS\_CRYPTO.CHAIN\_CBC  + DBMS\_CRYPTO.PAD\_PKCS5; ENCRYPTION\_KEY RAW(16) := UTL\_RAW.CAST\_TO\_RAW('dddden3546'); SALT\_RAW RAW(2000) := UTL\_RAW.CAST\_TO\_RAW('saltys'); |

Листинг 6.1 – Данные необходимые для шифрования и дешифрования

Функция ENCRYPT, представленная в листинге 6.2, принимает строку PLAINTEXT в качестве аргумента и возвращает зашифрованные данные в формате RAW. Для зашифровки данных используется функция DBMS\_CRYPTO.ENCRYPT с помощью выбранного алгоритма шифрования и ключа шифрования. В данном пакете мы используем алгоритм AES128 в режиме шифрования CBC, который обеспечивает дополнительную защиту от атак на шифрованные данные. Функция принимает на вход строку, которую необходимо зашифровать, а также параметры ENCRYPTION\_TYPE, ENCRYPTION\_KEY и SALT\_RAW, определенные в пакете. Строка преобразуется в RAW формат с помощью функции UTL\_RAW.CAST\_TO\_RAW, а ключ шифрования формируется путем конкатенации ENCRYPTION\_KEY и SALT\_RAW. Результат шифрования сохраняется в переменную и возвращается функцией.

|  |
| --- |
| FUNCTION ENCRYPT(P\_PLAINTEXT VARCHAR2) RETURN RAW DETERMINISTIC  IS  ENCRYPTED\_RAW RAW(2000);  KEY\_RAW RAW(2000) := UTL\_RAW.CONCAT(ENCRYPTION\_KEY, SALT\_RAW); BEGIN  ENCRYPTED\_RAW := DBMS\_CRYPTO.ENCRYPT  (  SRC => UTL\_RAW.CAST\_TO\_RAW(P\_PLAINTEXT),  TYP => ENCRYPTION\_TYPE,  KEY => KEY\_RAW  );  RETURN ENCRYPTED\_RAW; END ENCRYPT; |

Листинг 6.2 – Процедура шифрования ENCRYPT

Функция DECRYPT, представленная в листинге 6.3, принимает зашифрованные данные в формате RAW и возвращает дешифрованные данные в виде строки VARCHAR2. для расшифровки данных, зашифрованных с помощью функции DBMS\_CRYPTO.ENCRYPT используется функция DBMS\_CRYPTO.DECRYPT. Она принимает на вход зашифрованные данные в формате RAW, а также параметры ENCRYPTION\_TYPE, ENCRYPTION\_KEY и SALT\_RAW, определенные в пакете. Ключ шифрования формируется аналогично функции DBMS\_CRYPTO.ENCRYPT. Затем функция DBMS\_CRYPTO.DECRYPT используется для расшифровки данных с помощью ENCRYPTION\_TYPE и KEY\_RAW. Результат расшифровки сохраняется в переменную, которая затем преобразуется в строку с помощью функции UTL\_RAW.CAST\_TO\_VARCHAR2 и возвращается функцией.

|  |
| --- |
| FUNCTION DECRYPT(P\_ENCRYPTEDTEXT RAW) RETURN VARCHAR2 DETERMINISTIC  IS  DECRYPTED\_RAW RAW(2000);  KEY\_RAW RAW(2000) := UTL\_RAW.CONCAT(ENCRYPTION\_KEY, SALT\_RAW); BEGIN  DECRYPTED\_RAW := DBMS\_CRYPTO.DECRYPT  (  SRC => P\_ENCRYPTEDTEXT,  TYP => ENCRYPTION\_TYPE,  KEY => KEY\_RAW  );  RETURN (UTL\_RAW.CAST\_TO\_VARCHAR2(DECRYPTED\_RAW)); END DECRYPT; |

Листинг 6.3 – Процедура дешифрования DECRYPT

Таким образом, мы используем пакет криптографических функций DBMS\_CRYPTO для защиты данных в базе данных Oracle12с, добавляя триггеры, которые используют созданные нами функции шифрования и дешифрования, для шифрования и обновления значений в столбцах LOGIN и PASSWORD таблиц CUSTOMERS и EMPLOYEES.

## Маскирование данных

Маскирование является важным методом защиты конфиденциальных данных от несанкционированного доступа в базе данных. мы сосредоточимся на методе редактирования и, в частности, на политиках маскирования для таблиц EMPLOYEES и CUSTOMERS.

Техника редактирования заключается в селективном скрытии конфиденциальных данных от несанкционированных пользователей путем замены их нечувствительным значением или частичным маскированием конфиденциальных данных. В нашем случае мы используем пакет DBMS\_REDACT для реализации политик редактирования.

Начнем с политики редактирования для таблицы EMPLOYEES. Политика называется REDACT\_CUSTOMER и применяется к столбцу PHONE\_NUMBER. Эта политика реализована с помощью функции типа REGEXP, которая применяет регулярное выражение для поиска и замены конфиденциальных данных. Используется регулярное выражение '\d+(\d{4})$', которое соответствует любой последовательности цифр, заканчивающейся четырьмя цифрами. Найденные данные заменяются на '\*\*\*\*\1', что заменяет найденные цифры на четыре звездочки, за которыми следуют последние четыре цифры. Политика REDACT\_CUSTOMER также определена для применения ко всем вхождениям найденных данных в начале строки.

Кроме столбца PHONE\_NUMBER, политика редактирования также применяется к столбцу EMAIL в таблице EMPLOYEES. Эта политика также реализована с помощью функции типа REGEXP, с использованием другого регулярного выражения. Шаблон '(^.)(.)(@.)$' соответствует первому символу строки, за которым следует любое количество символов, а затем символ '@' и любые символы, которые следуют. Найденные данные заменяются на '\1' || RPAD('x', LENGTH('\2'), 'x') || '\3', что заменяет средние символы на то же количество символов 'x'.

Последняя политика замещения для таблицы EMPLOYEES применяется к столбцу SALARY. Для этой политики используется тип функции FULL, который полностью заменяет конфиденциальные данные на нечувствительное значение. Политика определена для применения ко всем вхождениям конфиденциальных данных в столбце.

Те же политики замещения также реализованы для таблицы CUSTOMERS, с теми же регулярными выражениями и значениями замены.

Для управления применением этих политик замещения мы используем выражение 'SYS\_CONTEXT(''SYS\_SESSION\_ROLES'',''C##PLATINUM\_KITCHEN\_MASKING'') = ''FALSE'''. Это выражение указывает, что политики замещения должны применяться только тогда, когда текущий пользователь не имеет роли C##PLATINUM\_KITCHEN\_MASKING.

В заключение, политики замещения представленные в приложение З, реализованные для таблиц EMPLOYEES и CUSTOMERS, используют типы функций REGEXP и FULL для селективного скрытия конфиденциальных данных от неавторизованных пользователей. Политики управляются выражением, которое указывает, когда политики должны применяться. Эти политики являются важным аспектом безопасности базы данных и помогают защитить конфиденциальные данные от несанкционированного доступа.

# Краткое описание приложения для демонстрации

## Пример работы приложения от лица менеджера

Когда пользователь открывает приложение, первым экраном, который ему показывается, является окно входа, которое представлено на рисунке 7.1. Здесь пользователь может ввести свой логин и пароль для авторизации в системе. После успешной авторизации система автоматически определяет роль пользователя и предоставляет ему соответствующие возможности в зависимости от роли, которую он имеет.

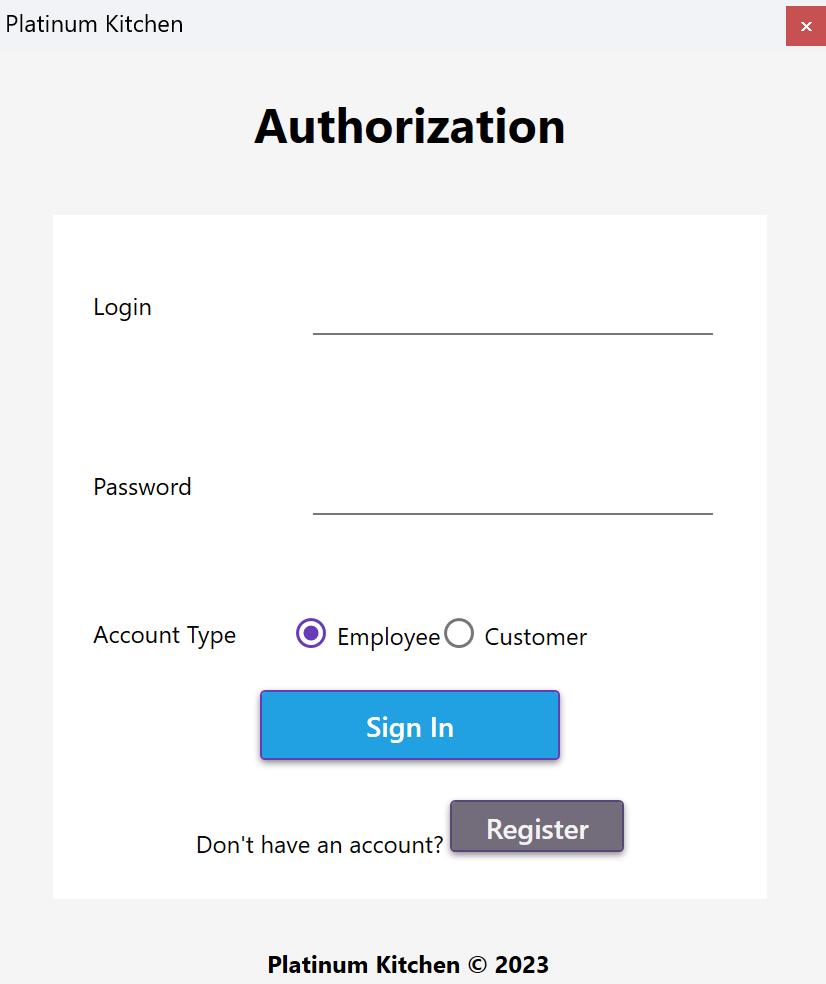


Рисунок 7.1 – Авторизация в приложении

После успешной авторизации в приложении ресторана, менеджеру доступны различные возможности для управления рестораном. Он может добавлять, удалять и многое другое.

В частности, одной из возможностей, которые предоставляются менеджеру, является резервация столика для посетителей ресторана, представленная на рисунке 7.2. Для этого менеджер может использовать соответствующую функцию в приложении, где необходимо выбрать дату и время бронирования, количество гостей, а также столик, который будет зарезервирован. После того как резервация будет подтверждена, система автоматически заблокирует выбранный столик на заданное время, чтобы избежать дублирования бронирований. Это позволяет менеджеру эффективно управлять ресурсами ресторана и обеспечить комфортное пребывание гостей.

Приложение, предназначенное для использования в ресторане, было разработано с учетом потребностей каждой роли, что делает его максимально интуитивно понятным и удобным в использовании.

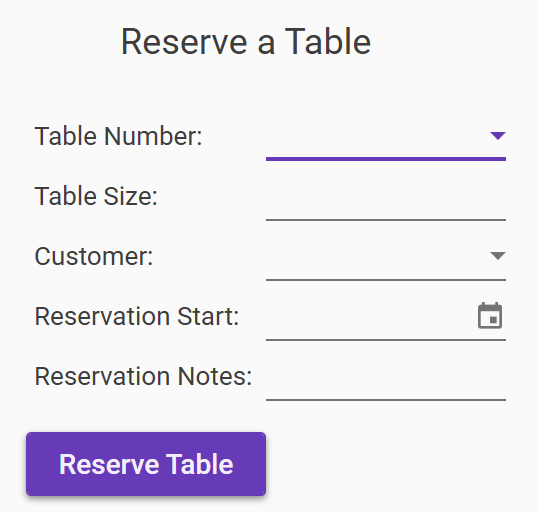


Рисунок 7.2 – Резервация столика

Благодаря такому подходу, пользователи могут без лишних затруднений взаимодействовать с приложением и осуществлять необходимые операции. Кроме того, важно отметить, что для обеспечения максимальной безопасности и защиты данных, были применены соответствующие механизмы, предотвращающие случайный доступ к конфиденциальной информации.

# Руководство пользователя

## Руководство использования приложения

После авторизации под ролью ADMIN или MANAGER, предоставляется доступ к админ-панели, её возможности представлены на рисунке 8.1

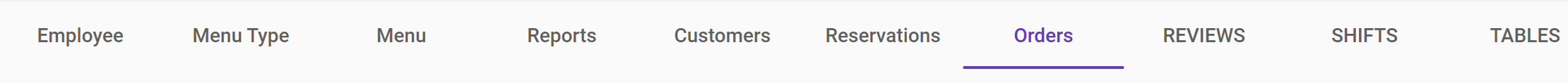


Рисунок 8.1 – Список возможностей админа

После чего в каждом пункте менеджер или администратор может найти необходимый функционал, предназначенный для этого пункта, для примера на рисунке 8.2 продемонстрирован функционал пункта Orders.

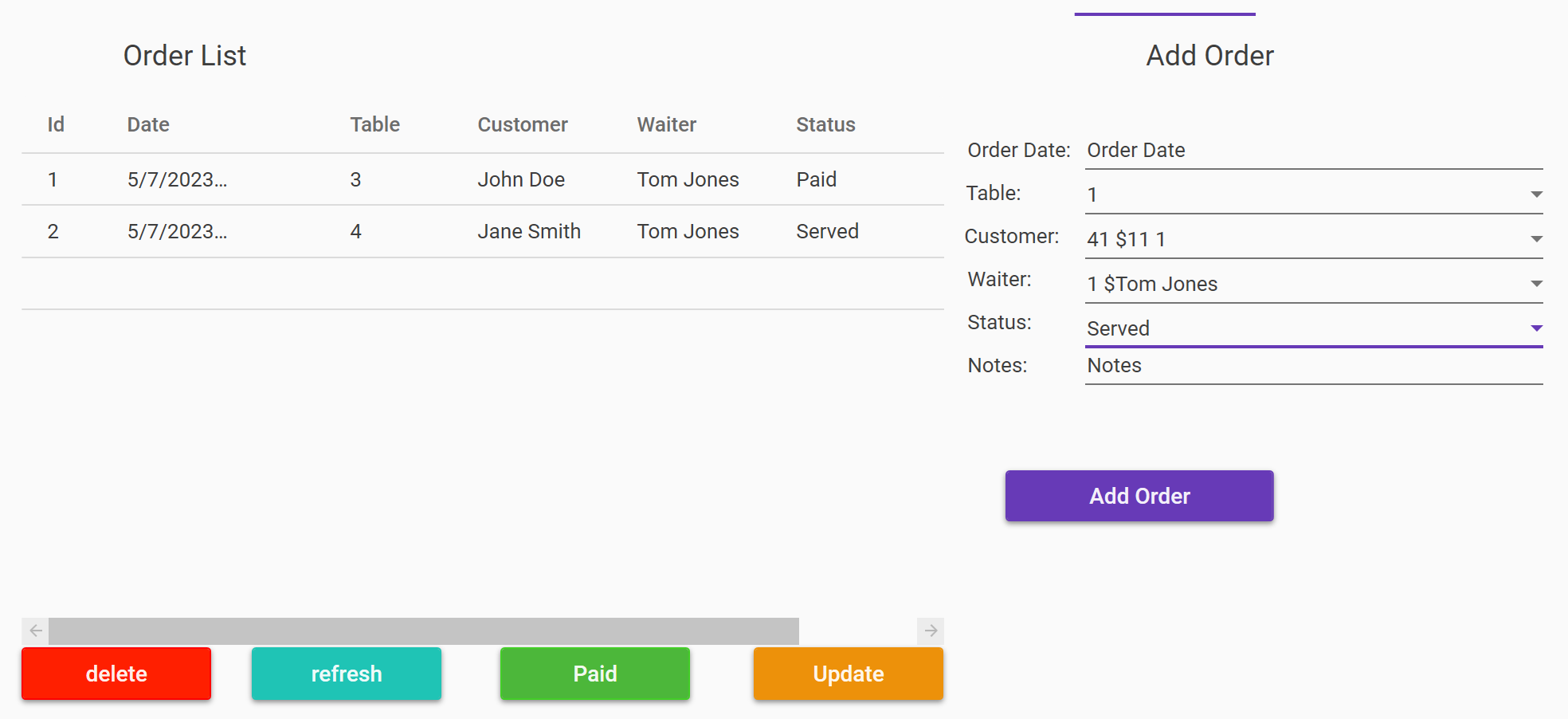


Рисунок 8.2 – Пункт Orders

Как видно на рисунке 8.2 пункт содержит достаточно действий для манипуляций с данными в рамках этого пункта.  
Что касательно ролей CLIENT, WAITER, EMPLOYEE у них есть как общие пункты, так и уникальные, например клиент, не может просматривать все заказы, а только свои.

# Заключение

В проекте была успешно реализована база данных ресторана, которая включает управление информацией о сотрудниках, меню, заказах, столиках и анализ выполненных заказов. Были применены разные возможности базы данных от простых операций вставки и удаления до написания сложных триггеров и процедур. Важным аспектом проекта было обеспечение безопасности данных с помощью технологии шифрования и маскирования. Была построена иерархия процедур доступа к данным, которая обеспечивает различный уровень доступа для разных пользователей, таких как администраторы, менеджеры, официанты, работники и клиенты. Это позволяет каждому пользователю получить доступ только к тем данным, которые ему необходимы. Было проведено тестирование производительности базы данных на большом объеме данных, что позволило оптимизировать ее структуру и проанализировать планы запросов к таблицам. Применение технологий базы данных в данной работе позволило значительно упростить и ускорить процессы управления данными ресторана.

# Список используемых источников

1. Маскировка данных в Oracle 12c [Электронный ресурс]. – Режим доступа: https://www.ktexperts.com/oracle-data-masking-12c/. – Дата доступа: 30.04.2023.
2. Шифрование данных в Oracle [Электронный ресурс]. – Режим доступа: https://www.nazmulhuda.info/encryption-decryption-using-dbms\_crypto. – Дата доступа: 30.04.2023.
3. Работа с файлами в Oracle [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/cd/F49540\_01/DOC/server.815/a68001/utl\_file.htm. – Дата доступа: 1.05.2023.
4. Работа с данными формата Xml в Oracle [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/19/arpls/DBMS\_XMLGEN.html. – Дата доступа: 2.05.2023.
5. Поставщик данных Oracle для .NET [Электронный ресурс]. – Режим доступа: https://www.oracle.com/cis/database/technologies/appdev/dotnet/odp.html. – Дата доступа: 28.04.2023.

# Приложение А Листинг создания таблиц

|  |
| --- |
| create table EMPLOYEES  (  ID NUMBER not null  primary key,  FIRST\_NAME VARCHAR2(50),  LAST\_NAME VARCHAR2(50),  LOGIN VARCHAR2(50),  PASSWORD VARCHAR2(50),  POSITION VARCHAR2(50),  EMAIL VARCHAR2(100),  PHONE\_NUMBER VARCHAR2(20),  SALARY NUMBER(10, 2)  )  create table CUSTOMERS  (  ID NUMBER not null  primary key,  FIRST\_NAME VARCHAR2(50),  LAST\_NAME VARCHAR2(50),  LOGIN VARCHAR2(50),  PASSWORD VARCHAR2(50),  EMAIL VARCHAR2(100),  PHONE\_NUMBER VARCHAR2(20)  )  create table TABLES  (  ID NUMBER not null  primary key,  TABLE\_NUMBER VARCHAR2(10),  TABLE\_SIZE VARCHAR2(20),  TABLE\_STATUS VARCHAR2(20)  )  create table MENUTYPE  (  ID NUMBER not null  primary key,  NAME VARCHAR2(50)  )  create table MENU  (  ID NUMBER not null  primary key,  NAME VARCHAR2(50),  DESCRIPTION VARCHAR2(200),  PRICE NUMBER(10, 2),  MENU\_TYPE\_ID NUMBER  references MENUTYPE,  NOTES VARCHAR2(200)  )  create table ORDERS  (  ID NUMBER not null  primary key,  ORDER\_DATE TIMESTAMP(6),  TABLE\_ID NUMBER  references TABLES,  CUSTOMER\_ID NUMBER  references CUSTOMERS,  WAITER\_ID NUMBER  references EMPLOYEES,  STATUS VARCHAR2(50),  NOTES VARCHAR2(200)  )  create table ORDERITEMS  (  ID NUMBER not null  primary key,  ORDER\_ID NUMBER  references ORDERS,  MENU\_ID NUMBER  references MENU,  QUANTITY NUMBER,  NOTES VARCHAR2(200)  )  create table REVIEWS  (  ID NUMBER not null  primary key,  ORDER\_ID NUMBER  references ORDERS,  RATING NUMBER,  NOTES VARCHAR2(200)  )  create table SHIFTS  (  ID NUMBER not null  primary key,  EMPLOYEE\_ID NUMBER  references EMPLOYEES,  START\_TIME TIMESTAMP(6),  END\_TIME TIMESTAMP(6),  NOTES VARCHAR2(200)  ) |

# Приложение Б Листинг пакета для шифрования данных

|  |
| --- |
| create PACKAGE MyCryptoPackage  AS  FUNCTION ENCRYPT(P\_PLAINTEXT VARCHAR2) RETURN RAW DETERMINISTIC;  FUNCTION DECRYPT(P\_ENCRYPTEDTEXT RAW) RETURN VARCHAR2 DETERMINISTIC;  END;  /  create PACKAGE BODY MyCryptoPackage  AS  ENCRYPTION\_TYPE PLS\_INTEGER := DBMS\_CRYPTO.ENCRYPT\_AES128  + DBMS\_CRYPTO.CHAIN\_CBC  + DBMS\_CRYPTO.PAD\_PKCS5;  ENCRYPTION\_KEY RAW(16) := UTL\_RAW.CAST\_TO\_RAW('dddden3546');  SALT\_RAW RAW(2000) := UTL\_RAW.CAST\_TO\_RAW('saltys');  FUNCTION ENCRYPT(P\_PLAINTEXT VARCHAR2) RETURN RAW DETERMINISTIC  IS  ENCRYPTED\_RAW RAW(2000);  KEY\_RAW RAW(2000) := UTL\_RAW.CONCAT(ENCRYPTION\_KEY, SALT\_RAW);  BEGIN  ENCRYPTED\_RAW := DBMS\_CRYPTO.ENCRYPT  (  SRC => UTL\_RAW.CAST\_TO\_RAW(P\_PLAINTEXT),  TYP => ENCRYPTION\_TYPE,  KEY => KEY\_RAW  );  RETURN ENCRYPTED\_RAW;  END ENCRYPT;  FUNCTION DECRYPT(P\_ENCRYPTEDTEXT RAW) RETURN VARCHAR2 DETERMINISTIC  IS  DECRYPTED\_RAW RAW(2000);  KEY\_RAW RAW(2000) := UTL\_RAW.CONCAT(ENCRYPTION\_KEY, SALT\_RAW);  BEGIN  DECRYPTED\_RAW := DBMS\_CRYPTO.DECRYPT  (  SRC => P\_ENCRYPTEDTEXT,  TYP => ENCRYPTION\_TYPE,  KEY => KEY\_RAW  );  RETURN (UTL\_RAW.CAST\_TO\_VARCHAR2(DECRYPTED\_RAW));  END DECRYPT;  END; |

# Приложение В Листинг создания триггеров

|  |
| --- |
| create trigger ENCRYPT\_CUSTOMER\_DATA\_INSERT  before insert  on CUSTOMERS  for each row  when (new.login IS NOT NULL AND new.password IS NOT NULL)  DECLARE  tmp\_id NUMBER;  BEGIN  LOOP  tmp\_id := CUSTOMERS\_SEQ.NEXTVAL;  SELECT COUNT(\*) INTO tmp\_id FROM CUSTOMERS WHERE id = tmp\_id;  EXIT WHEN tmp\_id = 0;  END LOOP;  :NEW.id := CUSTOMERS\_SEQ.currval;  :NEW.login := UTL\_RAW.CAST\_TO\_VARCHAR2(MyCryptoPackage.ENCRYPT(:NEW.login));  :NEW.password := UTL\_RAW.CAST\_TO\_VARCHAR2(MyCryptoPackage.ENCRYPT(:NEW.password));  END;  create trigger ENCRYPT\_CUSTOMER\_DATA\_UPDATE  before update of LOGIN,PASSWORD  on CUSTOMERS  for each row  when (new.login IS NOT NULL AND new.password IS NOT NULL)  BEGIN  IF :NEW.login != :OLD.login THEN  :NEW.login := UTL\_RAW.CAST\_TO\_VARCHAR2(MyCryptoPackage.ENCRYPT(:NEW.login));  END IF;  IF :NEW.password != :OLD.password THEN  :NEW.password := UTL\_RAW.CAST\_TO\_VARCHAR2(MyCryptoPackage.ENCRYPT(:NEW.password));  END IF;  END;  create trigger ENCRYPT\_EMPLOYEES\_DATA\_INSERT  before insert  on EMPLOYEES  for each row  when (new.login IS NOT NULL AND new.password IS NOT NULL)  DECLARE  tmp\_id NUMBER;  BEGIN  LOOP  tmp\_id := EMPLOYEES\_SEQ.NEXTVAL;  SELECT COUNT(\*) INTO tmp\_id FROM EMPLOYEES WHERE id = tmp\_id;  EXIT WHEN tmp\_id = 0;  END LOOP;  :NEW.id := EMPLOYEES\_SEQ.currval;  :NEW.login := UTL\_RAW.CAST\_TO\_VARCHAR2(MyCryptoPackage.ENCRYPT(:NEW.login));  :NEW.password := UTL\_RAW.CAST\_TO\_VARCHAR2(MyCryptoPackage.ENCRYPT(:NEW.password));  END;  create trigger ENCRYPT\_EMPLOYEES\_DATA\_UPDATE  before update of LOGIN,PASSWORD  on EMPLOYEES  for each row  when (new.login IS NOT NULL AND new.password IS NOT NULL)  BEGIN  IF :NEW.login != :OLD.login THEN  :NEW.login := UTL\_RAW.CAST\_TO\_VARCHAR2(MyCryptoPackage.ENCRYPT(:NEW.login));  END IF;  IF :NEW.password != :OLD.password THEN  :NEW.password := UTL\_RAW.CAST\_TO\_VARCHAR2(MyCryptoPackage.ENCRYPT(:NEW.password));  END IF;  END; |

# Приложение Г Листинг процедуры для общего импорта информации

|  |
| --- |
| create PROCEDURE import\_all\_xml\_data  IS  v\_clob CLOB;  BEGIN  FILE\_TO\_CLOB('TABLES', v\_clob);  IMPORT\_TABLES\_XML(v\_clob);  commit;  FILE\_TO\_CLOB('CUSTOMERS', v\_clob);  IMPORT\_CUSTOMERS\_XML(v\_clob);  commit;  FILE\_TO\_CLOB('EMPLOYEES', v\_clob);  IMPORT\_EMPLOYEE\_XML(v\_clob);  commit;  FILE\_TO\_CLOB('MENUTYPE', v\_clob);  IMPORT\_MENU\_TYPE\_XML(v\_clob);  commit;  FILE\_TO\_CLOB('MENU', v\_clob);  IMPORT\_MENU\_XML(v\_clob);  commit;  FILE\_TO\_CLOB('ORDERS', v\_clob);  IMPORT\_ORDERS\_XML(v\_clob);  commit;  FILE\_TO\_CLOB('REVIEWS', v\_clob);  IMPORT\_REVIEWS\_XML(v\_clob);  commit;  FILE\_TO\_CLOB('SHIFTS', v\_clob);  IMPORT\_SHIFTS\_XML(v\_clob);  commit;  FILE\_TO\_CLOB('ORDERITEMS', v\_clob);  IMPORT\_ORDER\_ITEMS\_XML(v\_clob);  commit;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' || SQLERRM);  END;  / |

# Приложение Д Листинг процедуры для заполнения таблиц

|  |
| --- |
| create PROCEDURE Populate\_All\_Tables AS  BEGIN  FOR i IN 1..100000  LOOP  Add\_Menu\_Type(TO\_CHAR(DBMS\_RANDOM.STRING('A', 10)));  Add\_Menu\_Item(DBMS\_RANDOM.STRING('A', 10),DBMS\_RANDOM.STRING('A', 100),  ROUND(DBMS\_RANDOM.VALUE(1, 100), 2),ROUND(DBMS\_RANDOM.VALUE(1, i)),  DBMS\_RANDOM.STRING('A', 100));  add\_employee(DBMS\_RANDOM.STRING('A', 10), DBMS\_RANDOM.STRING('A', 10),  DBMS\_RANDOM.STRING('A', 10), DBMS\_RANDOM.STRING('A', 10),  CASE ROUND(DBMS\_RANDOM.VALUE(1, 3))  WHEN 1 THEN 'Manager'  WHEN 2 THEN 'Waiter'  WHEN 3 THEN 'Chef'  END,  DBMS\_RANDOM.STRING('A', 10) || '@example.com',  ROUND(DBMS\_RANDOM.VALUE(1000, 10000), 2),  ROUND(DBMS\_RANDOM.VALUE(1000, 10000), 2));  add\_table(TO\_CHAR(i), ROUND(DBMS\_RANDOM.VALUE(1, 10)));  add\_customer(DBMS\_RANDOM.STRING('A', 10), DBMS\_RANDOM.STRING('A', 10),  DBMS\_RANDOM.STRING('A', 10), DBMS\_RANDOM.STRING('A', 10),  DBMS\_RANDOM.STRING('A', 10) || '@example.com',  DBMS\_RANDOM.STRING('N', 10));  Add\_Order(p\_table\_id => ROUND(DBMS\_RANDOM.VALUE(1, i)), p\_customer\_id => ROUND(DBMS\_RANDOM.VALUE(1, i)), p\_order\_date => SYSDATE,  p\_notes => DBMS\_RANDOM.STRING('A', 100),  p\_waiter\_id => ROUND(DBMS\_RANDOM.VALUE(1, i)), p\_status => CASE ROUND(DBMS\_RANDOM.VALUE(1, 4))  WHEN 1 THEN 'Reserved'  WHEN 2 THEN 'Served'  WHEN 3 THEN 'Open'  WHEN 4 THEN 'Paid'  END);  add\_order\_item(ROUND(DBMS\_RANDOM.VALUE(1, i)), ROUND(DBMS\_RANDOM.VALUE(1, i)),  ROUND(DBMS\_RANDOM.VALUE(1, 10)),  DBMS\_RANDOM.STRING('A', 100));  add\_shift(ROUND(DBMS\_RANDOM.VALUE(1, i)),  TRUNC(SYSDATE) + ROUND(DBMS\_RANDOM.VALUE(0, 23) / 24, 2),  TRUNC(SYSDATE) + ROUND(DBMS\_RANDOM.VALUE(24, 47) / 24, 2),  DBMS\_RANDOM.STRING('A', 100));  add\_review(ROUND(DBMS\_RANDOM.VALUE(1, i)),  ROUND(DBMS\_RANDOM.VALUE(1, 5)), DBMS\_RANDOM.STRING('A', 100));  if i mod 1000 = 0  then  DBMS\_OUTPUT.PUT\_LINE(i / 1000);  end if;  END LOOP;  END;  / |

# Приложение Е Листинг процедуры извлечение информации из файла

|  |
| --- |
| create PROCEDURE FILE\_TO\_CLOB  (  p\_file\_name IN VARCHAR2,  p\_clob OUT CLOB  )  AS  v\_file UTL\_FILE.FILE\_TYPE;  v\_filename VARCHAR2(100);  v\_buffer VARCHAR2(32767);  BEGIN  v\_filename := p\_file\_name || '.xml';  v\_file := UTL\_FILE.FOPEN('DATA\_DIR', v\_filename, 'r');  LOOP  UTL\_FILE.GET\_LINE(v\_file, v\_buffer);  IF v\_buffer = '</ROWSET>' THEN  p\_clob := p\_clob || v\_buffer;  EXIT;  ELSE  p\_clob := p\_clob || v\_buffer;  END IF;  END LOOP;  UTL\_FILE.FCLOSE(v\_file);  END;  / |

# Приложение Ж Листинг процедуры получения информации из файла

|  |
| --- |
| create PROCEDURE import\_tables\_xml  (  p\_file IN VARCHAR2  )  AS  v\_xml XMLTYPE := XMLTYPE(p\_file);  BEGIN  FOR item IN (  SELECT extractvalue(value(r), '/ROW/ID') AS id,  extractvalue(value(r), '/ROW/TABLE\_NUMBER') AS table\_number,  extractvalue(value(r), '/ROW/TABLE\_SIZE') AS table\_size,  extractvalue(value(r), '/ROW/TABLE\_STATUS') AS table\_status  FROM TABLE (XMLSEQUENCE(EXTRACT(v\_xml, '/ROWSET/ROW'))) r  )  LOOP  add\_table(  p\_table\_number => item.table\_number,  p\_table\_size => item.table\_size,  p\_table\_status => item.table\_status  );  END LOOP;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' || SQLERRM);  END;  / |

# Приложение З Листинг политик маскирования

|  |
| --- |
| BEGIN  DBMS\_REDACT.ADD\_POLICY(  object\_schema => 'C##PLATINUM\_KITCHEN\_ADMIN',  object\_name => 'EMPLOYEES',  column\_name => 'PHONE\_NUMBER',  policy\_name => 'REDACT\_CUSTOMER',  function\_type => DBMS\_REDACT.REGEXP,  regexp\_pattern => '\d+(\d{4})$',  regexp\_replace\_string => '\*\*\*\*\1',  regexp\_position => DBMS\_REDACT.RE\_BEGINNING,  regexp\_occurrence => DBMS\_REDACT.RE\_ALL,  expression => 'SYS\_CONTEXT(''SYS\_SESSION\_ROLES'',''C##PLATINUM\_KITCHEN\_MASKING'') = ''FALSE'''  );  DBMS\_REDACT.ALTER\_POLICY(  object\_schema => 'C##PLATINUM\_KITCHEN\_ADMIN',  object\_name => 'EMPLOYEES',  column\_name => 'EMAIL',  policy\_name => 'REDACT\_CUSTOMER',  function\_type => DBMS\_REDACT.REGEXP,  regexp\_pattern => '(^.)(.\*)(@.\*)$',  regexp\_replace\_string => '\1' || RPAD('x', LENGTH('\2'), 'x') || '\3',  expression => 'SYS\_CONTEXT(''SYS\_SESSION\_ROLES'',''C##PLATINUM\_KITCHEN\_MASKING'') = ''FALSE''',  action => DBMS\_REDACT.ADD\_COLUMN  );  DBMS\_REDACT.ALTER\_POLICY(  object\_schema => 'C##PLATINUM\_KITCHEN\_ADMIN',  object\_name => 'EMPLOYEES',  column\_name => 'SALARY',  policy\_name => 'REDACT\_CUSTOMER',  function\_type => DBMS\_REDACT.FULL,  expression => 'SYS\_CONTEXT(''SYS\_SESSION\_ROLES'',''C##PLATINUM\_KITCHEN\_MASKING'') = ''FALSE''',  action => DBMS\_REDACT.ADD\_COLUMN  );  END;  /  BEGIN  DBMS\_REDACT.ADD\_POLICY(  object\_schema => 'C##PLATINUM\_KITCHEN\_ADMIN',  object\_name => 'CUSTOMERS',  column\_name => 'PHONE\_NUMBER',  policy\_name => 'REDACT\_CUSTOMER',  function\_type => DBMS\_REDACT.REGEXP,  regexp\_pattern => '\d+(\d{4})$',  regexp\_replace\_string => '\*\*\*\*\1',  regexp\_position => DBMS\_REDACT.RE\_BEGINNING,  regexp\_occurrence => DBMS\_REDACT.RE\_ALL,  expression => 'SYS\_CONTEXT(''SYS\_SESSION\_ROLES'',''C##PLATINUM\_KITCHEN\_MASKING'') = ''FALSE'''  );  DBMS\_REDACT.ALTER\_POLICY(  object\_schema => 'C##PLATINUM\_KITCHEN\_ADMIN',  object\_name => 'CUSTOMERS',  column\_name => 'EMAIL',  policy\_name => 'REDACT\_CUSTOMER',  function\_type => DBMS\_REDACT.REGEXP,  regexp\_pattern => '(^.)(.\*)(@.\*)$',  regexp\_replace\_string => '\1' || RPAD('x', LENGTH('\2'), 'x') || '\3',  expression => 'SYS\_CONTEXT(''SYS\_SESSION\_ROLES'',''C##PLATINUM\_KITCHEN\_MASKING'') = ''FALSE''',  action => DBMS\_REDACT.ADD\_COLUMN  );  END; |