AZƏRBAYCAN RESPUBLİKASI ELM VƏ TƏHSİL NAZİRLİYİ
AZƏRBAYCAN TEXNİKİ UNİVERSİTETİ

Əlyazma hüququnda

M651R

Bayramov Yunsur Cəlal oğlu
Səlimi Əhməd Şah Davud Şah oğlu
Dəmirov Eldar Vasiliy oğlu

Создание готовых Excel отчетов на основе SQL и PL/SQL запросов для банковской среды

MAGİSTR
DİSSERTASİYASI

060544-Kompüter mühəndisliyi
"Kompüter sistemləri və şəbəkələri"

Elmi rəhbər: *t.e.n. Kərimov Kərim Vahid oğlu*

BAKI – 2023

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Создание готовых отчетов в формате Excel на основе запросов SQL и PL/SQL является важной задачей в банковской сфере. Одним из ключевых аспектов эффективного управления данными в банковской структуре является возможность анализа и представления информации в удобной и понятной форме. В этом разделе будет представлено введение в работу по созданию готовых отчетов Excel на основе запросов SQL и PL/SQL для банковского сектора, включая актуальность темы, цель работы и научные результаты.

Актуальность выбранной темы. В современной банковской отрасли объемы данных растут с каждым днем, и возможность анализа и представления этих данных становится все более важной. Excel является одним из наиболее широко используемых инструментов отчетности и анализа данных. Однако получение консолидированной информации из баз данных требует использования запросов SQL и PL/SQL для извлечения и обработки данных. Поэтому создание готовых отчетов Excel на основе запросов SQL и PL/SQL имеет высокую актуальность в банковской сфере.

Цель работы. Целью данной работы является разработка методики создания готовых отчетов в формате Excel на основе запросов SQL и PL/SQL для банковского сектора. Основное внимание уделяется эффективному извлечению и обработке данных из базы данных, их анализу и представлению в удобном формате с помощью Excel. Работа также направлена на оптимизацию процесса отчетности для упрощения и ускорения процесса анализа данных.

Научные результаты: В ходе работы были достигнуты следующие научные результаты:

- Разработана методика создания готовых отчетов Excel на основе запросов SQL и PL/SQL для банковского сектора. Этот метод включает этапы извлечения и обработки данных с использованием запросов SQL

и PL/SQL, а также представления их в Excel с использованием инструментов форматирования, графиков и сводных таблиц.

- Изучены лучшие практики и советы по оптимизации запросов SQL и PL/SQL для повышения производительности и эффективности извлечения данных из базы данных.
- Было проведено сравнение различных подходов к созданию отчетов в Excel, включая использование встроенных функций Excel и использование макросов VBA (Visual Basic for Applications), для определения наиболее эффективного способа создания отчетов в конкретной банковской среде.
- Представлены примеры создания готовых отчетов Excel на основе запросов SQL и PL/SQL для различных банковских задач, таких как анализ доходов и расходов, отслеживание активности клиентов и оценка рисков.

ГЛАВА I

ПОНЯТИЕ И ТИПЫ БАЗ ДАННЫХ. ЯЗЫКИ БАЗ ДАННЫХ (SQL И PL\SQL). ИСПОЛЬЗОВАНИЕ И ПОЛЬЗА БАЗЫ ДАННЫХ, SQL И PL\SQL ОПЕРАТОРОВ В БАНКОВСКОЙ СФЕРЕ

1.1. Введение в базы данных

Система управления базами данных — это сбор программ, разрешающих распоряжаться данными. Эта система помогает таблицам понять, чего хотят пользователи и какие таблицы отвечают пользователям.

Базы данных могут помочь решить многие проблемы, с которыми не могут справиться папки и файлы Excel:

1. Базы данных могут сохранять много данных;
2. Базы данных могут помочь взять под защиту данные. Только определенные люди могут получить доступ к данным. Кроме того, вы можете установить ограничения на то, кто и как может получить доступ к данным, например, только для чтения или только для редактирования;
3. Базы данных могут помочь контролировать точность данных с помощью различных видов проверок;
4. Базы данных также позволяют многим людям манипулировать данными одновременно.

Так что же такое база данных?

База данных — это предопределенное строение для сохранения информации; это комплект информации, которая упорядоченно хранится в электронной форме.

1.1 (СУБД)

Типы баз данных:

- Иерархические
- ООБД
- Реляционные
- Ключ-значение
- Графовые
- Колоночные
- Сетевая

1.2. Примеры реляционных базы данных

MySQL

1.2 (MySQL)

MySQL — это система управления реляционными базами данных с раскрытым отправным кодом. Как и иные реляционные базы данных, MySQL сохраняет данные в таблицах, заключающихся из строк и столбцов. Пользователи могут определять, манипулировать, контролировать и запрашивать данные с помощью языка структурированных запросов, более известного как SQL.

PostgreSQL

1.3 (PostgreSQL)

Это бесплатный инструмент управления реляционными базами данных, который предлагает некоторые функции MySQL, PostgreSQL поддерживает

высокий уровень гибкости и производительности при работе с большими базами данных. Он подходит для пользователей, которым нужна высокая скорость чтения/записи и крупномасштабный анализ данных. Известные пользователи PostgreSQL включают Reddit, Skype и Instagram.

MariaDB

1.4 (MariaDB)

Maria DB изначально разрабатывалась сообществом как форк MySQL, когда она была приобретена Oracle; MariaDB основана на MySQL с поддержкой большего количества движков и измененными ограничениями на хранение данных; он быстрее, чем MySQL, и может обрабатывать как SQL, так и NoSQL в одной и той же базе данных. Известными пользователями MariaDB являются Google, Mozilla и Wikimedia Foundation.

SQLite

1.5 (SQLite)

SQLite имеет некоторые ограничения, так как предлагает только частичные триггеры, имеет ограниченную функциональность. Кроме того, SQLite ограничивает максимальный размер базы данных до 32 000 столбцов и 140 ТБ. Известные пользователи включают Google Chrome, Mozilla Firefox, Opera и Safari..

1.3. Пример нереляционных баз данных

NoSQL (не только SQL) — это категория, охватывающая системы управления базами данных, отличные от реляционных баз данных,

использующих язык SQL. Этот класс СУБД появился в 2000-х годах в попытке решить проблемы масштабируемости, сохранив при этом простоту и гибкость.

1.6 (nosql)

1.4. Введение в SQL

Это стандартный язык для хранения, управления и извлечения данных из баз данных. Вы можете использовать SQL для доступа к базам данных и управления ими. В 1986 году Американский национальный институт приборостроения (ANSI) и Международная организация по стандартизации (ISO) признали стандарт SQL. SQL может запрашивать базу данных. SQL может быть основан на данных. Записи могут быть добавлены в базу данных с помощью SQL. Записи базы данных обновляются с помощью SQL. База данных может удалять записи с помощью SQL. В базе данных SQL он может создавать новые таблицы.

Таблицы — это базы данных, используемые для хранения данных в СУБД. Таблица состоит из строк и столбцов и содержит записи для данных. все обслуживание базы данных имеет одну или несколько таблиц. У каждой таблицы есть имя (например, «Клиенты» или «Заказы»), которое помогает пользователям найти ее.

Каждая таблица разделена на более мелкие объекты, определяемые полями. Поле — это столбец таблицы, который используется для хранения данных о каждой записи в таблице. Каждая запись в таблице называется записью, иногда называемой строкой. В таблице запись представляет собой

горизонтальную сущность. В таблице столбец — это вертикальный объект, содержащий все данные, соответствующие одной серии строк.

В реляционной базе данных взаимодействие между таблицами происходит через ключ, который является наиболее значимым протоколом при заключении и связи между ними. Существует несколько различных типов ключей. Индекс или номер статьи, например, служащий первичным ключом.

Внешний ключ — это набор ссылок для соединения двух таблиц; каждое появление ключа обязательно должно соответствовать первичному ключу в другой таблице. Обычно базы данных используются для хранения текстовой информации, но они хранят не только текстовую информацию, такую как изображения и некоторые документы, но и другие виды информации, хранящиеся в видеофайлах. Затем, используя язык SQL, разработчик приложения манипулирует базой данных. Он записывает данные в базу данных и извлекает их из нее. Они используют разные операции над базовыми данными.

Некоторые из наиболее распространенных типов данных SQL:

1. INTEGER: тип данных INTEGER используется для хранения значений без десятичных разрядов. Например, вы можете использовать INTEGER для хранения таких значений, как 1, 10 или -5.
2. VARCHAR: Тип данных VARCHAR используется для хранения строк выбора. Он может вводить буквы, цифры и специальные символы. Например, вы можете использовать VARCHAR для хранения имени, адреса или описания. Например, «Джон Смит» или «123 Main Street».
3. DATE: Тип данных DATE используется для хранения дат. Он может хранить только без информации о времени. Например, вы можете использовать DATE для хранения таких значений, как «2023-05-31», представляющих 31 мая 2023 года.
4. DECIMAL/NUMERIC: тип данных DECIMAL или NUMERIC используется для хранения десятичных чисел с фиксированной точностью и масштабом. Обычно его используют для финансовых расчетов. Например, вы можете использовать DECIMAL для хранения таких значений, как 3, 14 или 99, 99.
5. BOOLEAN: тип данных BOOLEAN используется для хранения данных измерений, обычно true (истина) или false (ложь). Например, вы можете

использовать BOOLEAN для хранения истинных или ложных значений, чтобы указать состояние чего-либо.

6. BLOB: тип данных BLOB (большой двоичный объект), используемый для хранения больших двоичных данных, таких как изображения или файлы. Например, вы можете использовать BLOB для хранения изображения или документа PDF.

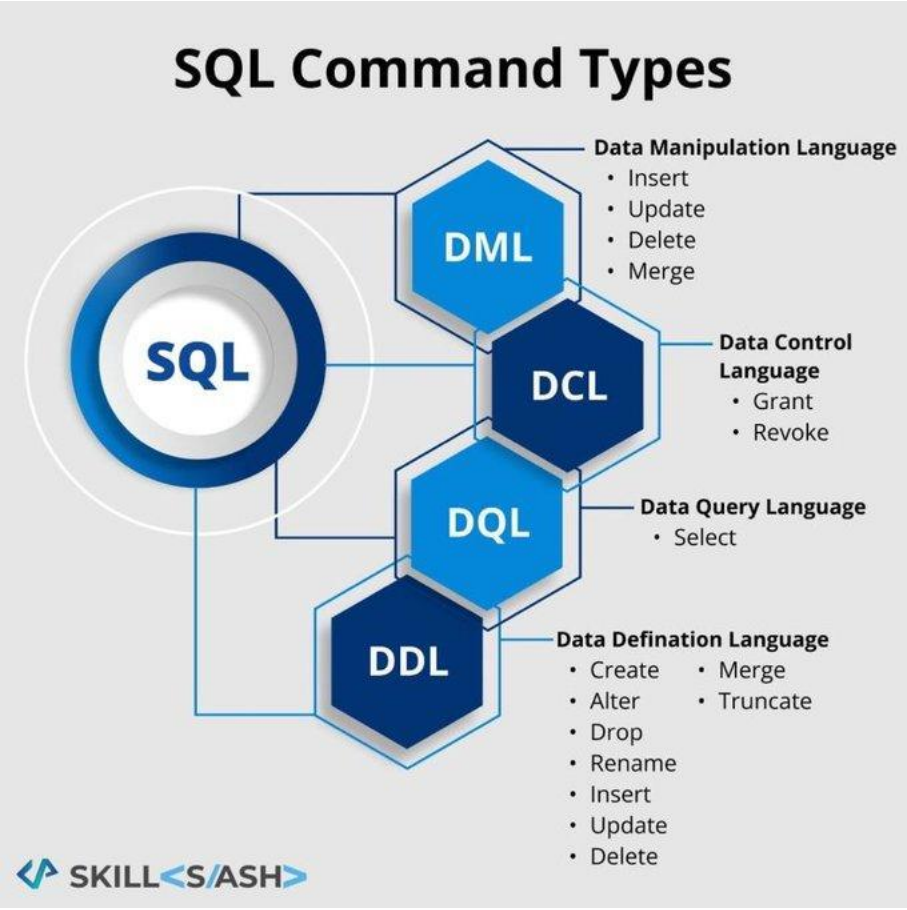
Некоторые из наиболее распространенных команд SQL включают:

1. SELECT: Команда SELECT используется для выбора данных из базы данных. Она позволяет извлечь информацию из таблицы и вернуть ее в результате запроса. Например, "SELECT * FROM users" вернет все строки и столбцы из таблицы "users".
2. INSERT: Команда INSERT используется для добавления новых записей в таблицу базы данных. Она позволяет вставить данные в таблицу. Например, "INSERT INTO users (name, age) VALUES ('John', 25)" добавит новую запись в таблицу "users" с именем "John" и возрастом 25.
3. UPDATE: Команда UPDATE используется для изменения существующих записей в таблице базы данных. Она позволяет обновить значения определенных столбцов в указанных строках. Например, "UPDATE users SET age = 30 WHERE name = 'John'" изменит возраст на 30 для записи с именем "John" в таблице "users".
4. DELETE: Команда DELETE используется для удаления записей из таблицы базы данных. Она позволяет удалить определенные строки, соответствующие указанным условиям. Например, "DELETE FROM users WHERE age > 30" удалит все записи из таблицы "users", где возраст больше 30.
5. CREATE: Команда CREATE используется для создания новых таблиц, баз данных или других объектов базы данных. Она позволяет создать новый объект с заданными характеристиками. Например, "CREATE TABLE users (id INT, name VARCHAR(50), age INT)" создаст новую таблицу "users" с тремя столбцами: "id" типа INT, "name" типа VARCHAR(50) и "age" типа INT.
6. JOIN: Команда JOIN используется для объединения данных из двух или более таблиц на основе определенных условий. Она позволяет связать данные из разных таблиц на основе общих значений столбцов. Например, "SELECT * FROM users JOIN orders ON users.id = orders.user_id" объединит данные из таблиц "users" и "orders" на основе значения столбца "id".

Обратите внимание, что вам следует адаптировать команды SQL к вашей конкретной базе данных и структуре таблиц.

1.4.1. Операторы SQL

Оператор - это конкретное слово или буква, которая используется для выполнения определенной операции и полезна при работе с данными, например, при выборе из коллекции на основе определенных параметров. Операторы SQL можно разделить на несколько групп в зависимости от их назначения.



1.7 (Операторы SQL)

DDL (Data Definition Language)

Операторы определения данных. Этот оператор можно использовать, когда нужно добавить в базу данных таблицу, содержащую новые данные, или, наоборот, удалить таблицу, содержащую неверные данные.

ALTER - Добавление, изменение или удаление столбцов.

CREATE - создание таблицы.

DROP - удаление объекта из базы данных.

DML (Data Manipulation Language)

Операторы манипулирования данными. Эти операторы работают со строками, атрибутами. Их можно использовать для внесения изменений в конкретные значения. Например, заменить поле в столбце "Фамилия" строки данными о сотруднике компании, который был женат или удалить строку, содержащую данные о сотруднике, который был уволен.

INSERT - Добавляет новые данные.

UPDATE - Изменение существующих данных.

SELECT - оператор запроса, который возвращает необходимую информацию.

DELETE - удаление строк из таблицы.

DCL (Data Control Language)

Это оператор, используемый для определения доступа к данным. Он определяет, кому разрешено отправлять запросы к базе данных и изменять объекты и значения. Например, вы можете отозвать доступ у человека, который перешел в другой отдел, или предоставить доступ к базе данных новому сотруднику отдела маркетинга или разработчику.

GRANT - Предоставить доступ к объекту.

REVOKE - отозвать ранее предоставленное право доступа.

DENY - отменяет разрешение, запрещая доступ.

TCL (Transaction Control Language)

Язык управления транзакциями. Транзакция - это набор команд, которые выполняются по одной за раз. Если все команды выполнены, транзакция проходит успешно, но если произошла ошибка, транзакция откатывается и все выполненные команды отменяются. Хорошим примером такой транзакции является онлайн-платеж, при котором банк сначала запрашивает сумму и получателя платежа, затем проверяет и подтверждает транзакцию, а затем запрашивает одноразовый код. На каждом этапе платеж может быть отменен, а транзакция откатана.

BEGIN - Определяет начало транзакции.

COMMIT - фиксирует изменения.

ROLLBACK - откат всех изменений.

Существуют различные версии языка SQL.Однако, чтобы соответствовать стандарту ANSI, по крайней мере, основные команды (SELECT, UPDATE, DELETE, INSERT, WHERE и т.д.) поддерживаются одинаково.

1.5. PL / SQL

PL/SQL (Procedural Language/Structured Query Language) - это язык программирования, который расширяет возможности SQL для создания более сложных и автоматизированных задач в базе данных Oracle.

PL/SQL используется для написания программ или скриптов, которые выполняются в базе данных Oracle. Он позволяет создавать процедуры, функции и блоки кода, которые могут быть выполнены на стороне сервера базы данных.

1.5.1. Преимущества PL/SQL

SQL является стандартным языком баз данных, а PL/SQL тесно интегрирован с SQL. PL/SQL позволяет одновременно отправлять в базу данных целый блок операторов. Это снижает сетевой трафик и обеспечивает высокую производительность приложений.PL/SQL дает программистам высокую производительность, поскольку он может запрашивать, преобразовывать и обновлять данные в базе данных.

PL/SQL экономит время на проектирование и отладку за счет мощных функций, таких как обработка исключений, инкапсуляция, сокрытие данных и объектно-ориентированные типы данных.Приложения, написанные на PL/SQL, полностью переносимы.PL/SQL обеспечивает высокий уровень безопасности.

Основной синтаксис

PL/SQL является языком с блочной структурой; это означает, что программы PL/SQL разделены и написаны в виде логических блоков кода. Каждый блок состоит из трех частей:

Декларации

Этот раздел начинается с ключевого слова DECLARE. Это необязательный раздел, в котором определяются все переменные, курсоры, подпрограммы и другие элементы, используемые в программе.

Исполняемые команды

Этот раздел заключен между ключевыми словами BEGIN и END и является обязательным. Он состоит из исполняемых операторов PL/SQL программы. Он должен иметь по крайней мере одну исполняемую строку кода, которая может быть просто командой NULL, чтобы указать, что ничего не должно выполняться.

1.5.2. Обработка исключений

Этот раздел начинается с ключевого слова EXCEPTION. Этот необязательный раздел содержит исключения, которые обрабатывают ошибки в программе.

Каждый оператор PL/SQL заканчивается точкой с запятой (;). Блоки PL/SQL могут быть вложены в другие блоки PL/SQL с помощью BEGIN и END. Ниже приведена базовая структура блока PL/SQL:

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling>
END;
```

PL/SQL также поддерживает условные операторы (IF-THEN-ELSE), циклы (FOR, WHILE), обработку исключений (EXCEPTION), работу с курсорами для обработки результатов запросов и многое другое. Он предоставляет более мощные возможности для автоматизации задач и управления данными в базе данных Oracle.

Важно отметить, что для выполнения PL/SQL кода вам может потребоваться использовать среду разработки, такую как Oracle SQL Developer или SQL*Plus, для создания и запуска программного кода PL/SQL.

1.6. Использование SQL и PL/SQL в банковской сфере.

SQL в банках.

SQL является стандартным языком запросов к базам данных и широко применяется в банковской сфере для выполнения следующих операций:

1.Запросы данных: Банки используют SQL для извлечения информации из базы данных, например, получение информации о клиентах, счетах, транзакциях и т.д. Запросы могут быть сложными, включая фильтрацию, сортировку, объединение таблиц и агрегацию данных.

2.Обновление данных: SQL позволяет вносить изменения в данные в базе данных банка. Это может быть изменение баланса счета, обновление персональных данных клиента или добавление новой транзакции. SQL операторы UPDATE, INSERT и DELETE используются для выполнения таких операций.

3.Управление структурой базы данных: Банки также используют SQL для управления структурой базы данных. Операторы CREATE, ALTER и DROP позволяют создавать, изменять и удалять таблицы, индексы и другие объекты базы данных. Это важно для обеспечения актуальности и эффективности базы данных.

PL/SQL в банках

PL/SQL - это расширение SQL, которое добавляет процедурные возможности к языку. В банковской сфере PL/SQL широко используется для следующих целей:

1.Хранимые процедуры и функции: Банки могут создавать хранимые процедуры и функции на PL/SQL для выполнения сложных операций и бизнес-логики. Например, процедура может быть создана для проверки валидности транзакции перед ее выполнением, расчета процентной ставки по счету или генерации отчетов.

2.Триггеры базы данных: PL/SQL триггеры могут быть использованы для автоматического выполнения определенных действий при определенных событиях в базе данных. В банковской сфере триггеры могут быть применены для регистрации и аудита изменений в данных, автоматического выполнения операций при определенных условиях и обеспечения целостности данных.

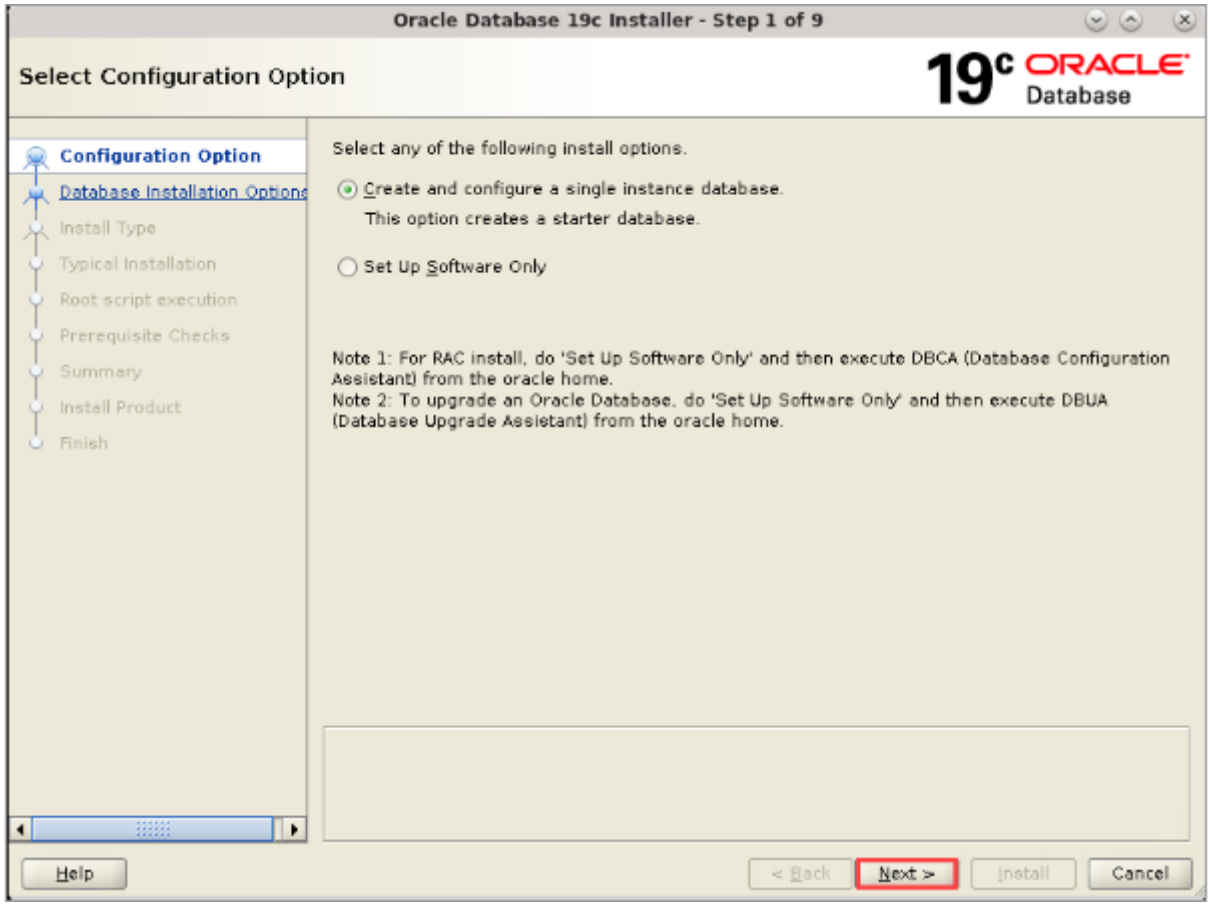
3.Обработка исключений: PL/SQL предоставляет механизм обработки исключений, который позволяет обрабатывать ошибки и ситуации исключений в коде. В банковской сфере это особенно важно, так как безопасность и надежность данных являются первостепенной задачей. Обработка исключений позволяет предотвратить потерю данных и обеспечить корректное выполнение операций.

1.7. Создание базы данных Oracle

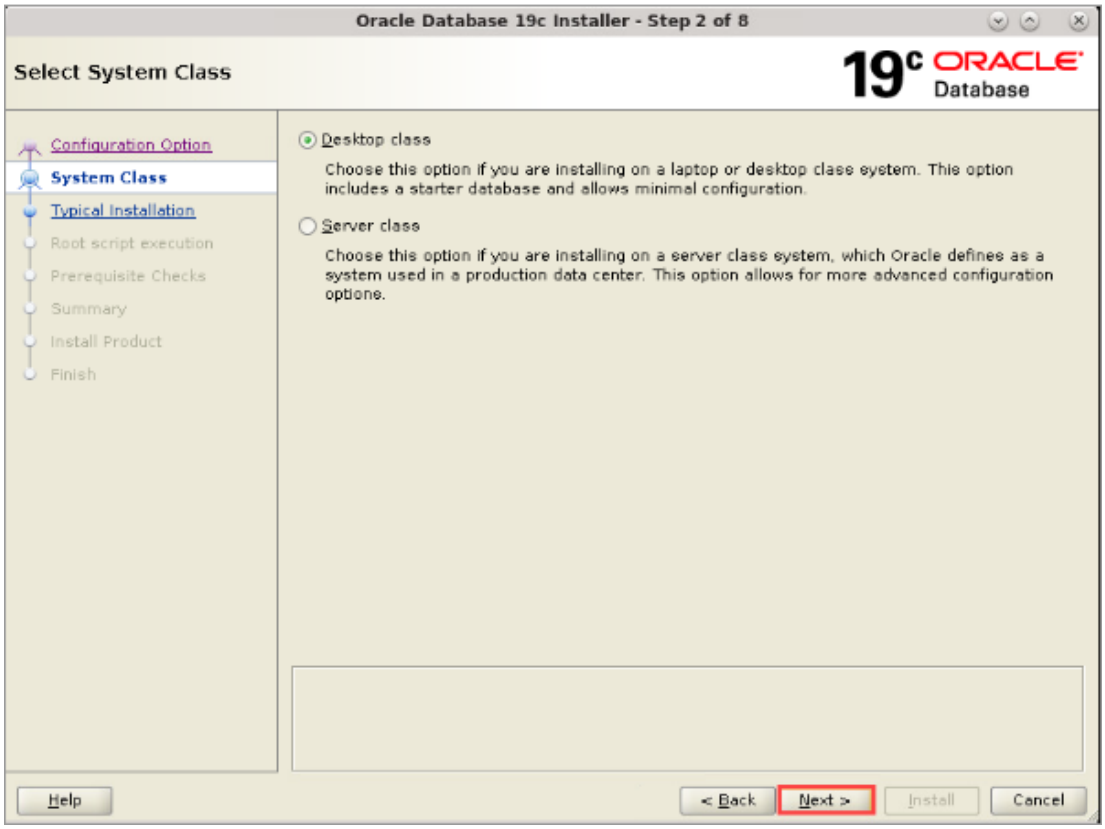
В нашей работе мы будем использовать базу данных Oracle. Мы выбрали эту базу данных ввиду её популярности и главное точности. Данная компания даёт 100% точность работоспособности. Также можем отметить, что данная база данных используется для больших объёмов данных, а так как в банковской среде имеются одни из самых больших объёмов данных логичнее использовать именно эту базу, Oracle. К примеру для небольших баз данных таких как веб-сайты обычно используют SQL SERVER. Приступим к созданию базы данных. Для начала нам нужно железо под нашу базу данных. В качестве железа мы используем обычный ноутбук так как у нас не будет больших объемов информации ведь цель работы заключается не в этом. Далее мы должны убедиться что на нашем устройстве имеется достаточно памяти для поднятии базы. Примерно нужно 6-7 гб для этого, но надо учесть, что в дальнейшем нам надо будет создавать таблцы, процедуры, функции и тому подобное, а также заполнять их соответствующими данными. Поэтому для перестраховки в нашей работе необходимо где-то 10 гб памяти на жестком диске. Базу данных Oracle можно создать в ходе процесса установки как в Windows, так и в UNIX-версиях. Универсальный инсталлятор Oracle предлагает несколько шаблонов для создания баз данных, в том числе шаблоны системы принятия решений (decision-support system — DSS) и оперативной обработки транзакций (online transaction processing — OLTP). Для создания базы данных можно вызвать также помощник по конфигурированию базы данных (Oracle Database Configuration Assistant — DBCA), который представляет собой средство с графическим интерфейсом пользователя, помогающее выполнять инсталляцию. Мы воспользуемся помощником для конфигурации, так как работать через командную строку SQL PLUS трата времени и нам не нужно такая детальная настройка базы.
Надо войти на свой компьютер в качестве члена административной группы, уполномоченной устанавливать программное обеспечение Oracle Database и создавать базу данных.

Если мы выполняете установку с дистрибутива, то надо дистрибутив Oracle Database в свой компьютер. Функция автозапуска автоматически открывает окно выбора продукта для установки. Если же мы загрузим программное обеспечение для установки с веб-сайта Oracle, то нам необходимо следовать инструкциям на сайте, чтобы вызвать универсальный установщик Oracle.

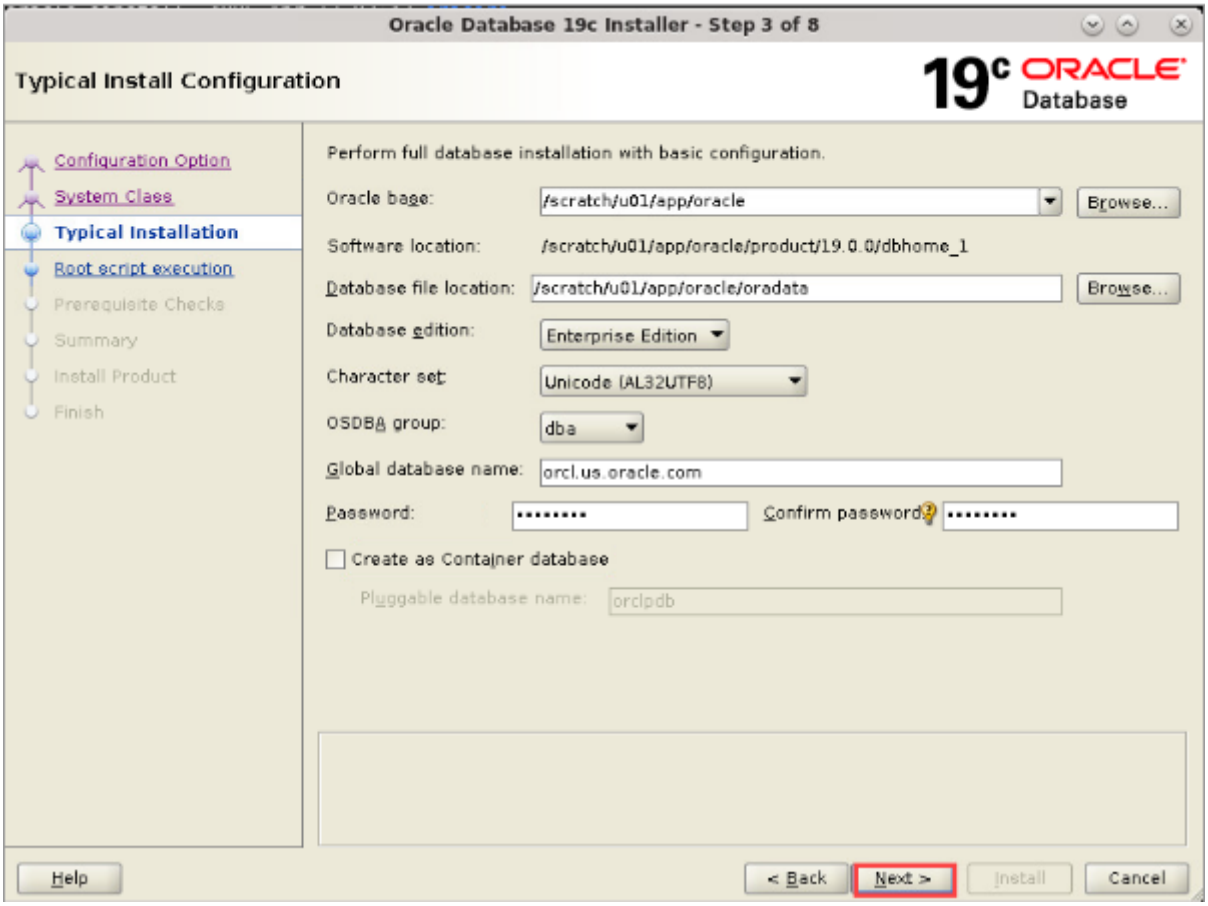
Появится окно выбора варианта конфигурации. Выберите «Создать и настроить базу данных с одним экземпляром». Жмем "Далее.



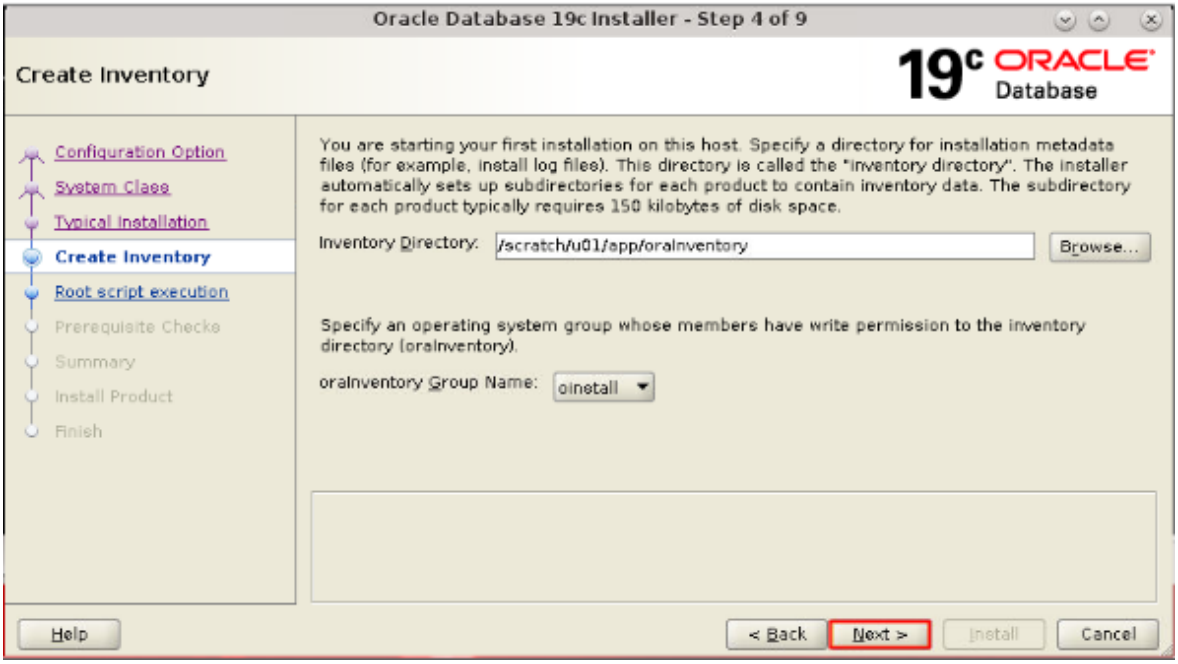
Появится окно класса системы. Выберем «Класс рабочего стола» и жмем «Далее».



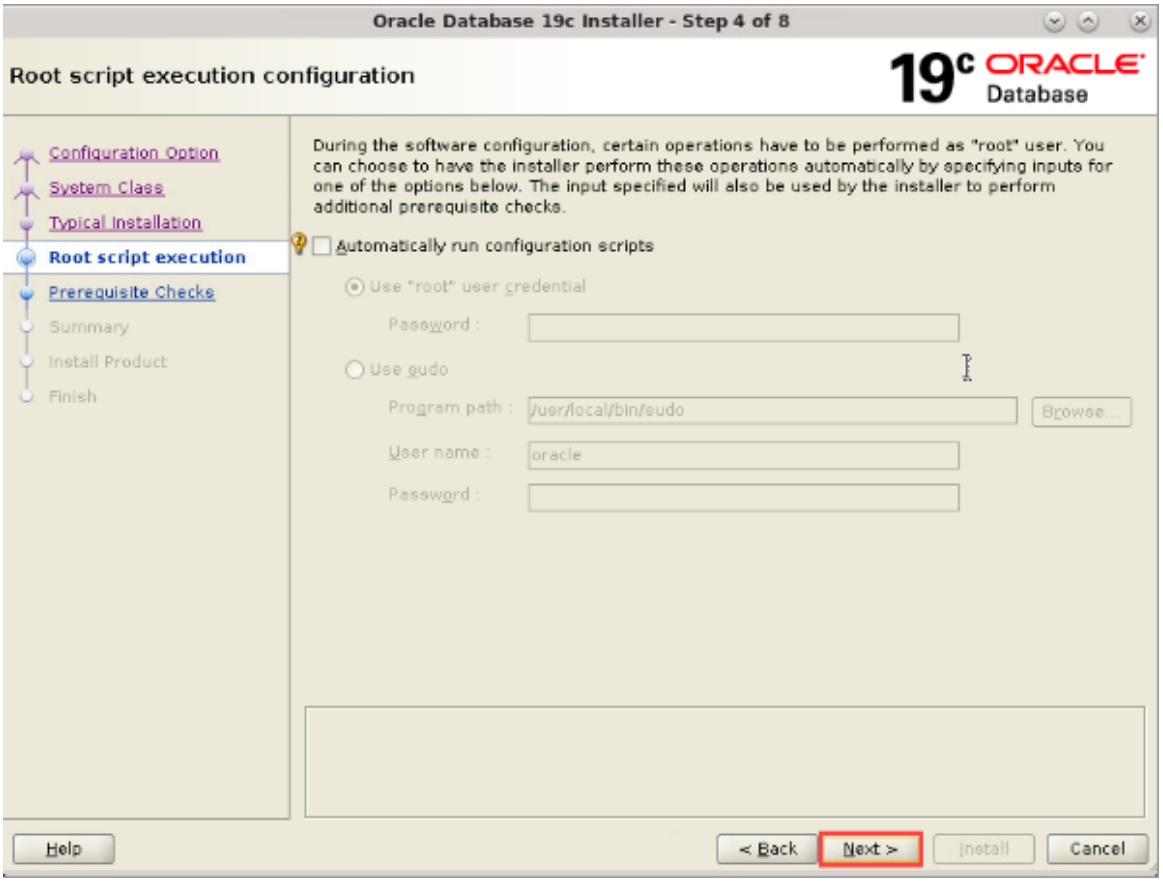
Появится окно Типовая конфигурация установки. В поле «Административный пароль» укажем пароль, который будет использоваться для учетных записей SYS, SYSTEM, SYSMAN и DBSNMP. (Предупреждение отображается, если пароль не соответствует стандартам паролей Oracle) Снимим флажок «Создать как контейнерную базу данных». Жмем "Далее.



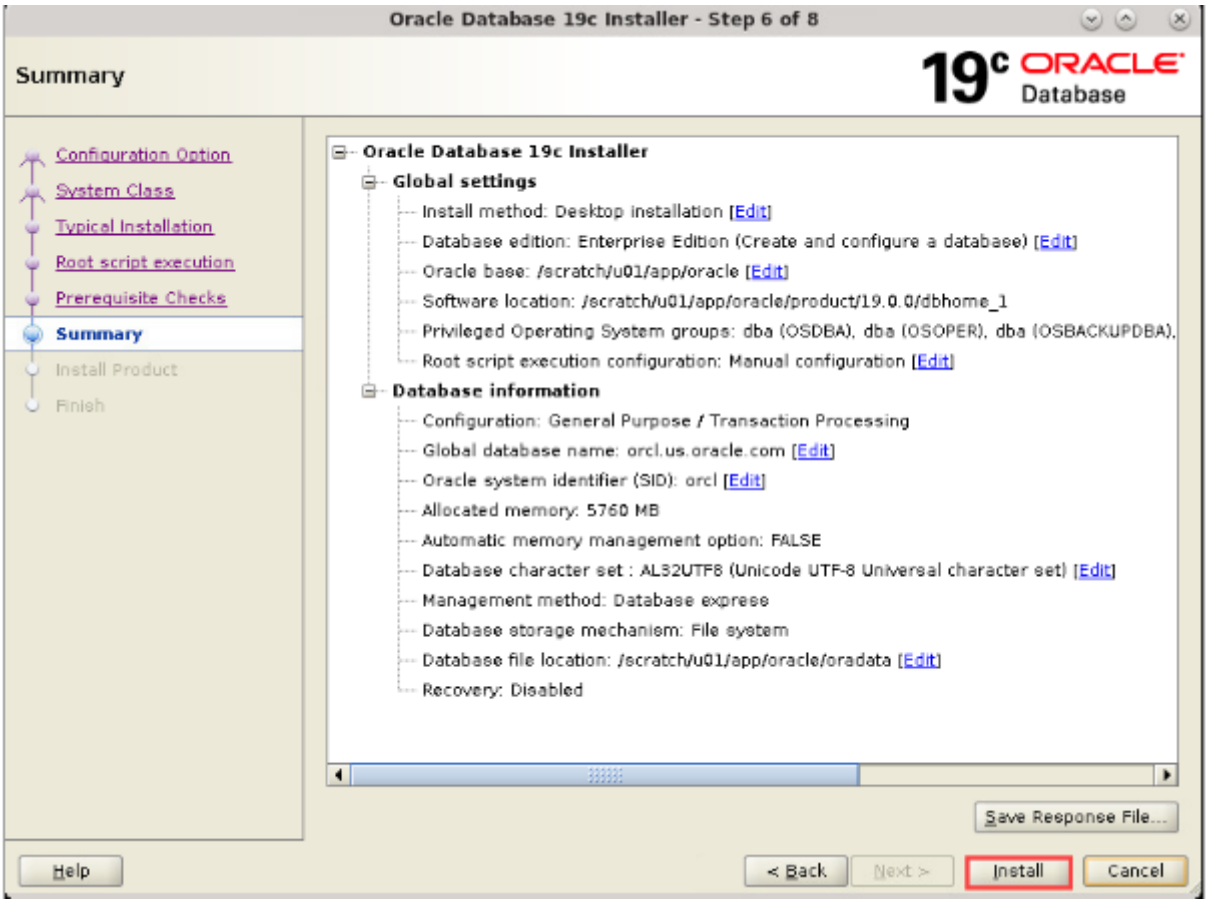
Появится окно создания инвентаря. Примем значения по умолчанию и жмем Далее.



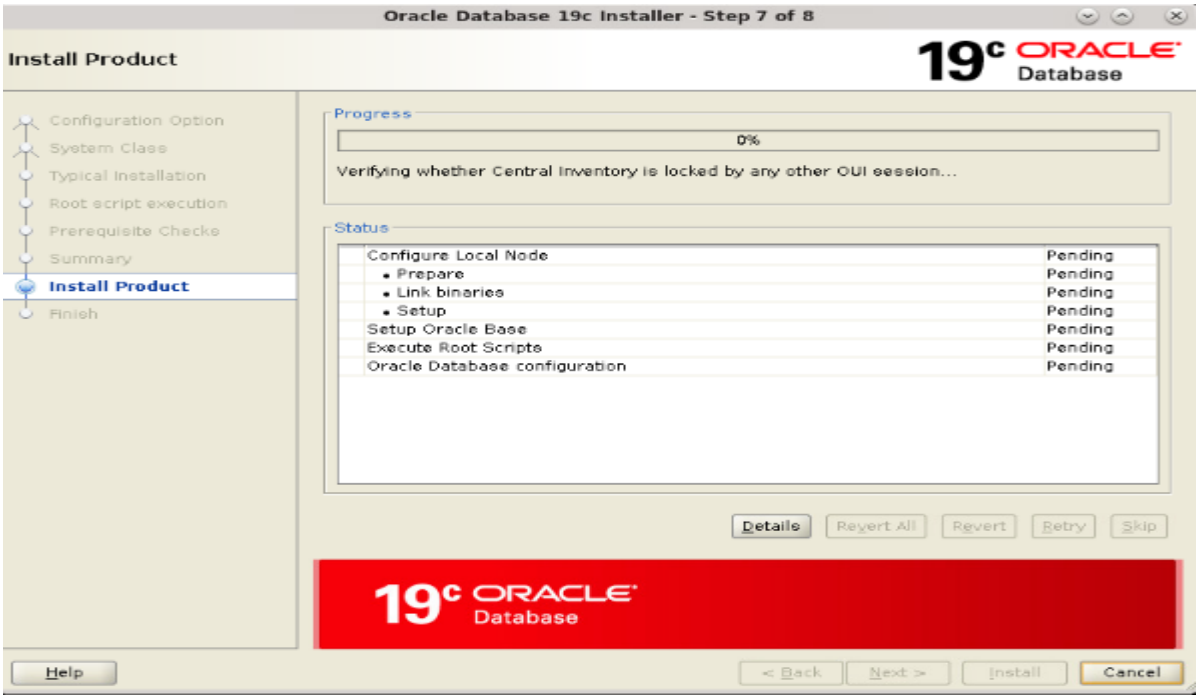
Появится окно выполнения корневого сценария. Только для операционных систем Linux и UNIX вам будет предложено запустить сценарии конфигурации. Для автоматического запуска корневых сценариев выберем Автоматически запускать сценарии конфигурации. Кроме того, мы можем запускать сценарии конфигурации вручную от имени пользователя root. В этом сценарии мы решили запускать сценарии конфигурации вручную. Жмем "Далее.



Выполняются предварительные проверки, и появляется окно Сводка. Проверяем настройки и жмем «Установить».



Появится окно прогресса.



После оконания установки база будет уже доступна. И мы можем подключиться к ней. Мы будем использовать для этого 2 инструмента: PL\SQL developer и TOAD.

ГЛАВА II

ПОДГОТОВКА МОДЕЛИ БАЗЫ ДАННЫХ ДЛЯ БАНКОВСКОЙ СТУРКТУРЫ

2.1. Анализ банковской структуры

Анализ банковской структуры является важным этапом, который позволяет понять организационную структуру и бизнес-процессы банка. Целью анализа является выявление основных сущностей, которые представляют интерес для базы данных. Это позволяет разработать эффективную модель базы
данных, которая отражает реальные потребности и операции банка. Например организация структуры таблиц и связей между ними, ограничения уникальностей, реализация продедур и т.д.

2.1.1. Идентификация основных сущностей

Анализ банковской структуры является первым шагом в процессе подготовки модели базы данных для банковской организации. В этом разделе будет рассмотрена важность анализа банковской структуры и методы идентификации основных сущностей, которые будут составлять основу базы данных.

Значение анализа банковской структуры

Анализ банковской структуры является важным этапом, который позволяет понять организационную структуру и бизнес-процессы банка. На основе этого анализа мы можем выявить что для базы данных банковской структуры является одним самым важных наличие таблицы для клиентов, счетов, а также тразакций. Это позволяет разработать эффективную модель базы данных, которая отражает реальные потребности и операции банка.

Идентификация основных сущностей

При анализе банковской структуры следующие основные сущности обычно выделяются:

1. Клиенты: Клиенты являются ключевой сущностью в банковской структуре. Это могут быть физические лица, юридические лица или организации. Идентификация клиентов включает атрибуты, такие как имя, адрес, контактная информация и другие данные, которые необходимы для идентификации и обслуживания клиентов.

2. Счета: Счета представляют финансовые отношения между клиентами и банком. Идентификация счетов включает атрибуты, такие как номер счета, тип счета, валюта, баланс и другие финансовые характеристики. Счета могут быть связаны с определенными клиентами или организациями.

3. Транзакции: Транзакции представляют финансовые операции, которые выполняются на счетах клиентов. Это могут быть депозиты, снятия наличных, переводы и другие операции. Идентификация транзакций включает атрибуты, такие как дата, время, тип транзакции, сумма и связанные счета.

4. Продукты и услуги: Банки предлагают различные продукты и услуги своим клиентам, такие как кредиты, дебетовые и кредитные карты, ипотеки и др. Идентификация продуктов и услуг включает атрибуты, такие как название продукта, условия предоставления, процентные ставки и другие характеристики.

5. Сотрудники: Сотрудники банка играют важную роль в банковской структуре. Идентификация сотрудников включает атрибуты, такие как имя, должность, контактная информация и другие данные, связанные с персоналом.

2.1.2. Определение атрибутов для каждой сущности

После идентификации основных сущностей в банковской структуре, следующим шагом является определение атрибутов для каждой сущности. Атрибуты представляют характеристики и свойства сущностей, которые необходимо учесть в базе данных. В этом разделе будет рассмотрено, какие

атрибуты могут быть определены для каждой основной сущности в банковской структуре.

Клиенты:

ClientID: идентификатор клиента (первичный ключ)

FirstName: имя клиента.

LastName: фамилия клиента

Address: физический адрес клиента.

PhoneNumber: номер телефона, электронная почта и другие контактные данные клиента.

DateOfBirth: дата рождения клиента.

```
-- Создание таблицы "Клиенты"
CREATE TABLE Clients (
    ClientID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DateOfBirth DATE,
    Address VARCHAR(100),
    PhoneNumber VARCHAR(20),
    Email VARCHAR(50)
);
```

Счета:

AccountID: идентификатор счета (первичный ключ)

ClientID: идентификатор клиента (вторичный ключ, который имееи связь с таблицей Clients со столбцом ClientID.

Account: кодовое название счета

Balance: текущий баланс на счете.

```
-- Создание таблицы "Счета"
CREATE TABLE Accounts (
    AccountID INT PRIMARY KEY,
    ClientID INT,
    Account VARCHAR(20),
    Balance DECIMAL(18, 2),
    FOREIGN KEY (ClientID) REFERENCES Clients(ClientID)
);
```

Транзакции:

TransactionID: идентификатор транзакции (первичный ключ)

AccountID: идентификатор счета (вторичный ключ, который связан с таблицей Accounts со столбцом AccountID

TransactionDate: дата и время выполнения транзакции.

TransactionType: тип операции, такой как депозит, снятие наличных, перевод или платеж.

Amount: сумма операции.

Также можно добавить следующие столбцы и еще много разновидностей столбцов, но для реализации нашей нет необходимости в этом.

Отправитель: связанный счет отправителя (если есть).

Получатель: связанный счет получателя (если есть).

Продукты и услуги:

Название продукта: название конкретного продукта или услуги.

Условия предоставления: основные условия и правила для использования продукта или услуги.

Процентные ставки: процентные ставки, связанные с продуктом или услугой.

Дополнительные характеристики: другие характеристики продукта или услуги, такие как льготы или ограничения.

```sql
-- Создание таблицы "Транзакции"
CREATE TABLE Transactions (
    TransactionID INT PRIMARY KEY,
    AccountID INT,
    TransactionType VARCHAR(20),
    Amount DECIMAL(18, 2),
    TransactionDate DATE,
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);
```

Сотрудники:

EmployeeID: идентификатор сотрудника

FirstName: имя сотрудника

LastName: фамилия сотрудника

Должность: должность или роль сотрудника в банке.

```
-- Создание таблицы "Сотрудники"
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DateOfBirth DATE,
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

2.2. Проектирование логической модели базы данных

2.2.1. Создание схемы сущность-связь (ER-диаграмма)

Проектирование логической модели базы данных является важным этапом в создании базы данных для банковской структуры. Одним из ключевых инструментов в проектировании логической модели является создание схемы сущность-связь (ER-диаграммы). В этом разделе будет рассмотрено, как создать ER-диаграмму для банковской структуры.

Создание схемы сущность-связь (ER-диаграмма)

Шаг 1: Идентификация сущностей

Первым шагом в создании ER-диаграммы является идентификация основных сущностей в банковской структуре, которые будут представлены в базе данных. Это могут быть клиенты, счета, транзакции, продукты и услуги, сотрудники и другие сущности, которые вы определили на предыдущих этапах анализа. Данный шаг мы уже начали делать в предыдущей подглаве.

Шаг 2: Определение атрибутов

Для каждой идентифицированной сущности определите их атрибуты, которые будут храниться в базе данных. Атрибуты представляют характеристики и свойства сущностей. Например, для сущности "Клиенты" атрибутами могут быть имя, адрес, контактная информация и другие данные, которые вы определили в предыдущем этапе.

Шаг 3: Определение связей

После определения сущностей и их атрибутов, определите связи между сущностями. Связи отображают отношения и взаимодействия между сущностями. Например, клиент может иметь один или несколько счетов, а

счет может быть связан с определенной транзакцией. Определите тип связей (один-ко-многим, многие-ко-многим и т. д.) и укажите их на диаграмме.

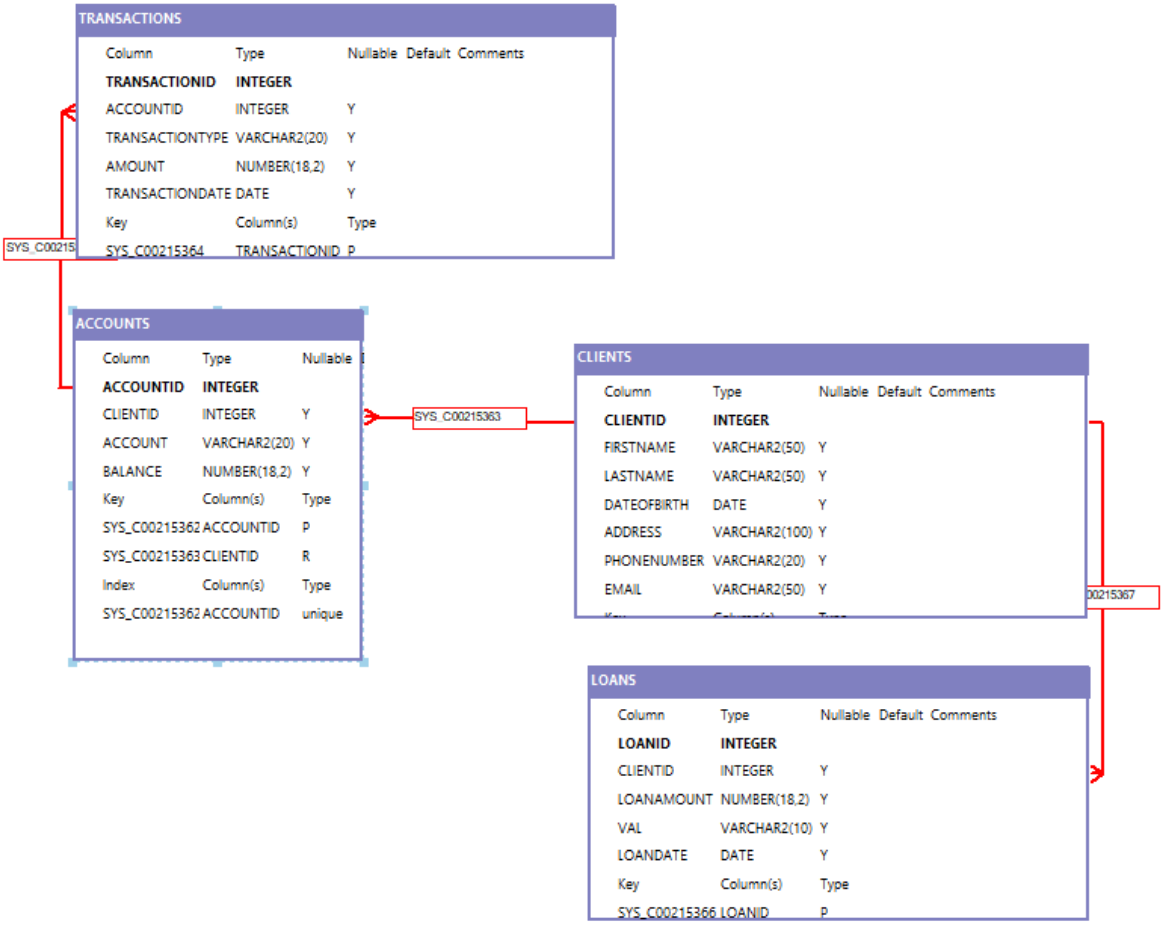
Например в нашем случае, связь между таблицами, которые мы создали в предыдущей главе обрзуется на основе следующей диаграммы. Мы решили их связать через идентификаторы. Этот метод является более оптимальным и надежным.

Шаг 4: Добавление связей и атрибутов на ER-диаграмму

Создайте ER-диаграмму, используя графические инструменты, такие как квадраты (для сущностей) и линии (для связей). Разместите сущности на диаграмме и соедините их линиями, указывая тип связей между ними. Добавьте атрибуты к каждой сущности, отображая их внутри квадратов.

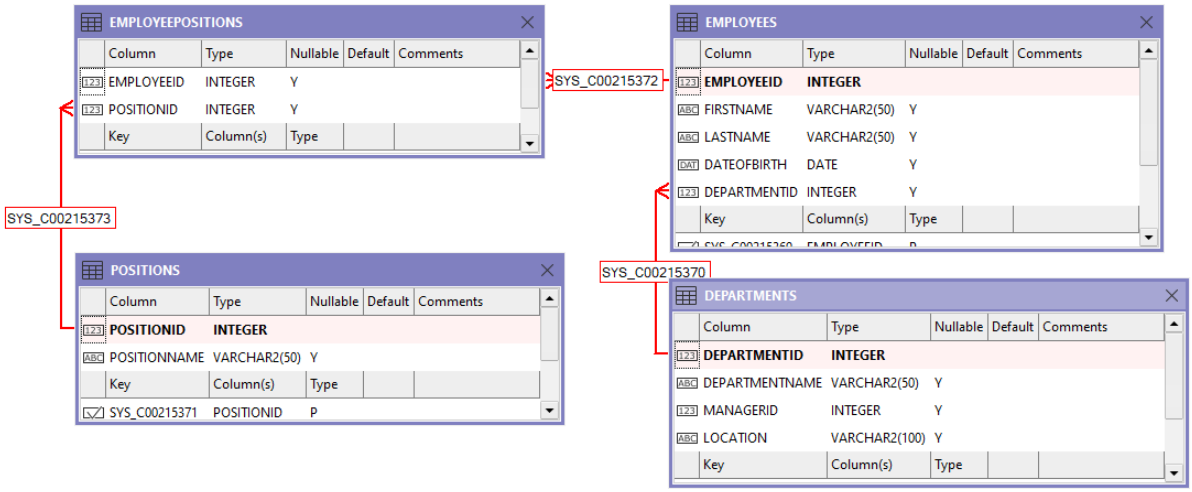
Шаг 5: Оптимизация и доработка ER-диаграммы

Оцените созданную ER-диаграмму и проведите оптимизацию, устраняя излишние связи или атрибуты, которые не являются существенными для модели. Проверьте, что все необходимые связи и атрибуты присутствуют на диаграмме и что она четко отражает бизнес-процессы и потребности банковской структуры.



Это связи между таблицами созданных для клиентов: клиенты, счета, транзакции, кредиты (один вид продукта)

Также мы еще придумали и создали таблицы для сотрудников банка, которые вы можете увидеть на следующей диаграмме. Отметим что связи также проводились через идентификаторы таблиц.



2.2.2. Нормализация данных

Что такое нормализация данных?

Нормализация данных - это процесс организации данных в базе данных таким образом, чтобы минимизировать избыточность, устранить аномалии и обеспечить эффективность при хранении и обработке данных. Нормализация следует набору правил, известных как нормальные формы, которые помогают структурировать данные и устранить проблемы, связанные с избыточностью и противоречиями.

Нормальные формы в нормализации данных

Первая нормальная форма (1НФ): Гарантирует, что каждый атрибут в таблице содержит только атомарные значения и нет повторений строк.

Вторая нормальная форма (2НФ): Устраняет избыточность данных, разбивая таблицу на отдельные сущности, чтобы каждая таблица содержала только атрибуты, которые зависят от полного первичного ключа.

Третья нормальная форма (3НФ): Гарантирует, что нет транзитивных зависимостей между атрибутами, то есть каждый атрибут зависит только от первичного ключа.

Четвертая нормальная форма (4НФ): Устраняет многозначные зависимости и многозначные зависимости относительно неключевых атрибутов.

Применение нормализации данных в банковской структуре

При проектировании логической модели базы данных для банковской структуры следует применять нормализацию данных для обеспечения оптимального хранения и обработки информации. Вот несколько примеров, как нормализация данных может быть применена в банковской структуре:

Разделение клиентов и счетов: Отдельные таблицы для клиентов и счетов помогут избежать избыточности данных и обеспечат более эффективное хранение и поиск информации о клиентах и их счетах.

Разделение транзакций и счетов: Отдельные таблицы для транзакций и счетов позволят эффективно отслеживать и хранить информацию о каждой транзакции, связанной с определенным счетом, и обеспечат целостность данных.

Разделение продуктов и услуг: Создание отдельных таблиц для продуктов и услуг позволит эффективно хранить информацию о каждом продукте или услуге, включая их характеристики и связи с клиентами или счетами.

Преимущества нормализации данных

Избегает избыточности данных: Нормализация помогает избежать повторений и избыточности информации, что уменьшает объем хранимых данных и обеспечивает целостность.

Улучшает производительность: Хорошо спроектированная нормализованная база данных обеспечивает быстрый доступ к данным и эффективные операции чтения и записи.

Упрощает обновление и модификацию данных: При нормализации данные структурируются таким образом, что изменение или обновление информации становится более простым и минимизирует риск ошибок.

2.3. Проектирование физической модели базы данных. Выбор типа базы данных. Определение структуры таблиц и полей

Мы используем базу данных Oracle. Мы выбрали ее так как она считается самой точной базой данных в мире. Сама компания даёт 100% точность работы.

Определение структуры таблиц и полей

После выбора типа базы данных необходимо определить структуру таблиц и полей для эффективного хранения данных. Вот несколько основных шагов для определения структуры таблиц и полей:

Идентификация сущностей: Определите основные сущности, которые будут представлены в базе данных, например, клиенты, счета, транзакции и т.д.

Определение полей: Для каждой сущности определите ее атрибуты или поля. Например, для сущности "клиент" могут быть определены поля, такие как имя, фамилия, адрес, контактная информация и т.д.

Установление связей: Определите связи между сущностями. Например, сущность "транзакция" может быть связана с сущностями "счет" и "клиент".

Определение первичных и внешних ключей: Установите первичные ключи для каждой таблицы, чтобы обеспечить уникальность идентификации записей. Определите внешние ключи для связей между таблицами.

Нормализация данных: Примените принципы нормализации данных, о которых было рассказано ранее, для устранения избыточности и обеспечения целостности данных.

2.4. Заполнение таблиц данными

В данной подглаве мы будем заполнять таблицы данными. Мы уже создали несколько таблиц, включая таблицы "Клиенты", "Счета", "Транзакции", "Кредиты", "Отделы", "Сотрудники", "Должности" и "Сотрудники-

Должности". Теперь мы подробно рассмотрим, как мы заполняем эти таблицы информацией.

Начнем с таблицы "Клиенты". В этой таблице мы храним информацию о наших клиентах. Мы можем использовать различные источники данных, например, данные, предоставленные самими клиентами или импортированные из других систем. Мы создаем SQL-запросы или используем PL/SQL-скрипты для добавления записей в таблицу "Клиенты" с помощью оператора INSERT.

Подобным образом, мы заполняем таблицу "Счета" информацией о банковских счетах клиентов. Мы также можем создать SQL-запросы или использовать PL/SQL-скрипты для добавления записей в таблицу "Счета" на основе имеющихся данных или взаимодействия с другими системами.

Для таблицы "Транзакции" мы записываем информацию о финансовых транзакциях, выполненных на счетах клиентов. Мы можем добавлять данные о транзакциях в таблицу в режиме реального времени или импортировать исторические данные из других источников. Для этого мы создаем SQL-запросы для вставки соответствующих записей.

Аналогичным образом мы заполняем таблицу "Кредиты" информацией о кредитах, предоставленных клиентам. Мы создаем новые кредитные договоры и добавляем соответствующие записи в таблицу или импортируем данные из других систем.

Для таблиц "Отделы", "Сотрудники", "Должности" и "Сотрудники-Должности" мы также используем SQL-запросы или PL/SQL-скрипты для добавления соответствующих записей в каждую таблицу на основе имеющихся данных о сотрудниках, должностях и их связях.

Таким образом, мы используем SQL и PL/SQL для заполнения таблиц данными в нашей банковской среде на основе созданных нами таблиц и соответствующих записей. Этот процесс позволяет нам создать полную базу данных с информацией о клиентах, счетах, транзакциях, кредитах, отделах, сотрудниках и должностях, которую мы можем использовать для дальнейшего анализа и создания готовых отчетов в Excel.

2.5. Обеспечение безопасности и доступа к данным

2.5.1. Разграничение доступа к данным

В банковской структуре обеспечение безопасности и контроль доступа к данным являются критически важными аспектами. Конфиденциальность, целостность и доступность данных являются основополагающими принципами, которые должны соблюдаться. В этом разделе будет рассмотрено, как обеспечить безопасность данных и реализовать разграничение доступа для эффективного управления информацией.

Идентификация и аутентификация пользователей

Первым шагом для обеспечения безопасности данных является идентификация и аутентификация пользователей. Каждому пользователю должен быть назначен уникальный идентификатор, и процесс аутентификации должен быть установлен для проверки подлинности пользователей перед предоставлением доступа к данным.

Разграничение доступа

Разграничение доступа - это процесс определения и установления различных уровней доступа к данным в зависимости от роли и полномочий пользователей. Это позволяет ограничить доступ к конфиденциальной информации только для соответствующих пользователей, минимизируя риск несанкционированного доступа.

Ролевая модель доступа

Одним из распространенных методов разграничения доступа является ролевая модель доступа. В этой модели определяются различные роли, связанные с разными уровнями доступа к данным. Пользователи назначаются определенным ролям, и доступ к данным предоставляется на основе этих ролей. Например, в банковской структуре могут быть определены роли, такие как администратор, менеджер, оператор и клиент, с разными уровнями доступа к данным.

Шифрование данных

Шифрование данных является важным аспектом обеспечения безопасности в банковской структуре. Он позволяет защитить конфиденциальную информацию путем преобразования данных в неразборчивую форму, которая может быть восстановлена только с использованием соответствующего ключа. Шифрование может применяться как при передаче данных, так и при их хранении.

Мониторинг и аудит доступа

Важным аспектом обеспечения безопасности и контроля доступа является мониторинг и аудит доступа к данным. Это включает в себя регистрацию и анализ активности пользователей, чтобы обнаружить любые несанкционированные попытки доступа или аномалии в использовании данных. Мониторинг и аудит помогают обеспечить отслеживаемость и ответственность в использовании данных.

2.5.2. Установка механизмов аутентификации и авторизации

Механизмы аутентификации

Механизмы аутентификации используются для проверки подлинности пользователей и установления их идентичности. В банковской среде могут быть использованы следующие механизмы аутентификации:

Парольная аутентификация: Пользователи вводят уникальный пароль для проверки подлинности. Важно использовать сильные пароли и принимать меры по их безопасному хранению, например, хэшированию.

Биометрическая аутентификация: Использование физических характеристик пользователей, таких как отпечатки пальцев, распознавание лица или голоса, для проверки подлинности.

Токены и карточки доступа: Использование физических или электронных устройств, содержащих уникальный идентификатор, для подтверждения подлинности.

Механизмы авторизации

Механизмы авторизации определяют, какие действия и ресурсы могут быть доступны пользователю после успешной аутентификации. В банковской среде может быть использовано несколько механизмов авторизации:

Ролевая модель доступа: Определение различных ролей с разными уровнями доступа к данным. Каждому пользователю назначается определенная роль, определяющая его полномочия и разрешенные операции.

Правила авторизации на основе политик безопасности: Установление набора правил и политик, определяющих, какие пользователи имеют доступ к конкретным данным или функциональности системы.

Ограничение доступа на уровне данных: Реализация ограничений доступа к конкретным данным на основе прав пользователей или групп пользователей.

Двухфакторная аутентификация

Двухфакторная аутентификация является дополнительным слоем безопасности и требует от пользователей предоставить два или более фактора для подтверждения своей идентичности. Например, комбинация пароля и одноразового кода, получаемого через SMS или мобильное приложение, повышает безопасность доступа к данным.

Журналирование и мониторинг

Журналирование и мониторинг активности пользователей являются важными механизмами для обеспечения безопасности данных. Регистрация и анализ логов позволяют выявить подозрительные действия и несанкционированный доступ к данным.

ГЛАВА III

СОЗДАНИЕ ПРОЦЕДУР И ПАКЕТОВ ДЛЯ РЕАЛИЗАЦИИ ЦЕЛИ ДИССЕРТАЦИИ

Целью данного исследования является разработка системы, которая позволит банкам генерировать готовые Excel отчеты на основе SQL и PL/SQL запросов. Задачи включают анализ требований банковской сферы к отчетам, проектирование системы генерации отчетов, разработку базы данных и модели данных, разработку SQL и PL/SQL запросов, разработку механизма генерации Excel отчетов, а также тестирование и оценку системы.

В этой главе рассматриваются практические аспекты диссертации с упором на реализацию результатов исследования. Основная цель — предоставить подробный отчет о шагах, связанных с созданием базы данных с помощью Oracle, включая создание таблиц, заполнение данными и разработку пакета PL/SQL, способного генерировать отчеты Excel на основе пользовательских запросов.

Практическая реализация диссертации играет решающую роль в проверке теоретических концепций, обсуждавшихся в предыдущих главах. Демонстрируя способность создавать функциональную базу данных, организовывать данные в таблицы, заполнять их релевантной информацией и создавать содержательные отчеты, эта глава демонстрирует практическую применимость и эффективность предлагаемого исследования.

3.1. Написание запроса SQL

3.1.1. SQL запрос, который будет возвращать нам список сотрудников банка.

Мы разработали запрос на SQL, который позволяет нам получить список сотрудников банка. Для этого мы используем несколько таблиц:

"employees" (сотрудники), "positions" (должности) и "employeepositions" (сотрудники-должности). Наш запрос выполняет следующие действия:

1. Мы выбираем все столбцы из таблицы "employees" и "positions" с помощью указания em. и p. соответственно.
2. Мы объединяем таблицы "employees", "positions" и "employeepositions" с помощью оператора FROM и указываем их сокращенные имена: em для "employees", p для "positions" и emp для "employeepositions".
3. Мы устанавливаем условия объединения таблиц: EM.EMPLOYEEID = emp.employeeid и P.POSITIONID = emp.positionid. Это гарантирует соответствие между идентификаторами сотрудников, должностями и связями сотрудники-должности.
4. У нас также есть условия фильтрации для должности (positionname) и идентификатора отдела (departmentid). Мы используем операторы LIKE для выполнения частичного сопоставления строк и операторы IS NULL для обработки возможности пустых значений переменных v_positionname и v_departmentid.

Пример использования запроса:

```
SELECT
   em.*,
   p.*
FROM
   employees em,
   positions p,
   employeepositions emp
WHERE
   EM.EMPLOYEEID = emp.employeeid
   AND P.POSITIONID = emp.positionid
```

```
AND (lower(positionname) LIKE '%' || lower('' || v_positionname || '') || '%' OR '' || v_positionname || '' IS NULL)
AND (departmentid LIKE '%' || ' || v_departmentid || ' || '%' OR ' || v_departmentid || ' IS NULL);
```

Этот запрос возвращает нам результат, состоящий из полной информации о сотрудниках и соответствующих должностях, учитывая фильтры по должности и отделу. Мы можем использовать этот запрос для получения требуемых данных о сотрудниках банка в нашей банковской среде. Результат от этого запроса будет выглядеть следующим образом.

CLIENTID	FIRSTNAME	LASTNAME	DATEOFBIRTH	ADDRESS
1	Abbasov	Pərviz	31.01.1966	AZ.1000, AZƏRBA
2	Yusifova	Mehriban	24.02.1973	AZ.1000, AZƏRBA
3	Ələsgərov	Namiq	05.05.1971	AZ1063, AZƏRBAY
4	Babayev	Baba	26.12.1979	AZ5000, AZƏRBAY
5	Ağayev	Murad	10.11.1965	AZ5000, AZƏRBAY
6	Abbasova	Dürdanə	07.07.1974	AZ5000, AZƏRBAY
7	Abdullayev	Arif	03.10.1990	AZ5000, AZƏRBAY
8	Bəşirov	Vasif	09.05.1985	AZ5000, AZƏRBAY
9	Ağakişiyev	Rövşən	28.10.1982	AZ5000, AZƏRBAY
10	Babayev	Seymur	12.08.1991	AZ5000, AZƏRBAY
11	Əsədov	Rəfail	15.02.1992	AZƏRBAYCAN, ABŞ
12	Moradı Asıabarakı	Esmaeıl	20.12.1982	AZƏRBAYCAN, ABŞ
13	Musayev	Rəhman	05.01.1970	AZƏRBAYCAN, AZƏ
14	Camalova	Şəlalə	30.12.1967	AZƏRBAYCAN, AĞD
15	Əhmədov	İlqar	11.03.1980	AZƏRBAYCAN, AĞD
16	Əhədov	Vüsal	10.05.1988	AZƏRBAYCAN, AĞS
17	Əhmədov	Toğrul	01.03.1982	AZƏRBAYCAN, AĞS
18	Yaqubov	Vüqar	26.07.1987	AZƏRBAYCAN, BAK
19	Şəfiyev	Ruslan	11.11.1981	AZƏRBAYCAN, BAK
20	Sarısu	Baycan	09.08.1971	AZƏRBAYCAN, BAK
21	Əliyev	Rəşad	26.11.1986	AZƏRBAYCAN, BAK
22	Əliyev	Rəşad	26.11.1986	AZƏRBAYCAN, BAK
23	Həsənov	Elmir	28.03.1971	AZƏRBAYCAN, BAK
24	Musayev	Fəzail	16.06.1975	AZƏRBAYCAN, BAK
25	Rasulova	Tamılla	29.05.1967	AZƏRBAYCAN, BAK
26	Səfərova	Şəhla	02.07.1982	AZƏRBAYCAN, BAK
27	Altinel	Miraç Baran	09.08.1978	AZƏRBAYCAN, BAK
28	Shafak	Nailə	29.11.1981	AZƏRBAYCAN, BAK
29	Həbibov	Mubariz	23.07.1968	AZƏRBAYCAN, BAK

3.1.2. SQL запрос, который будет возвращать нам список сотрудников банка.

Мы разработали SQL-запрос, который позволяет нам получить список клиентов банка на основе заданных фильтров. Для этого мы используем таблицу "CLIENTS" (Клиенты). Наш запрос выполняет следующие действия:

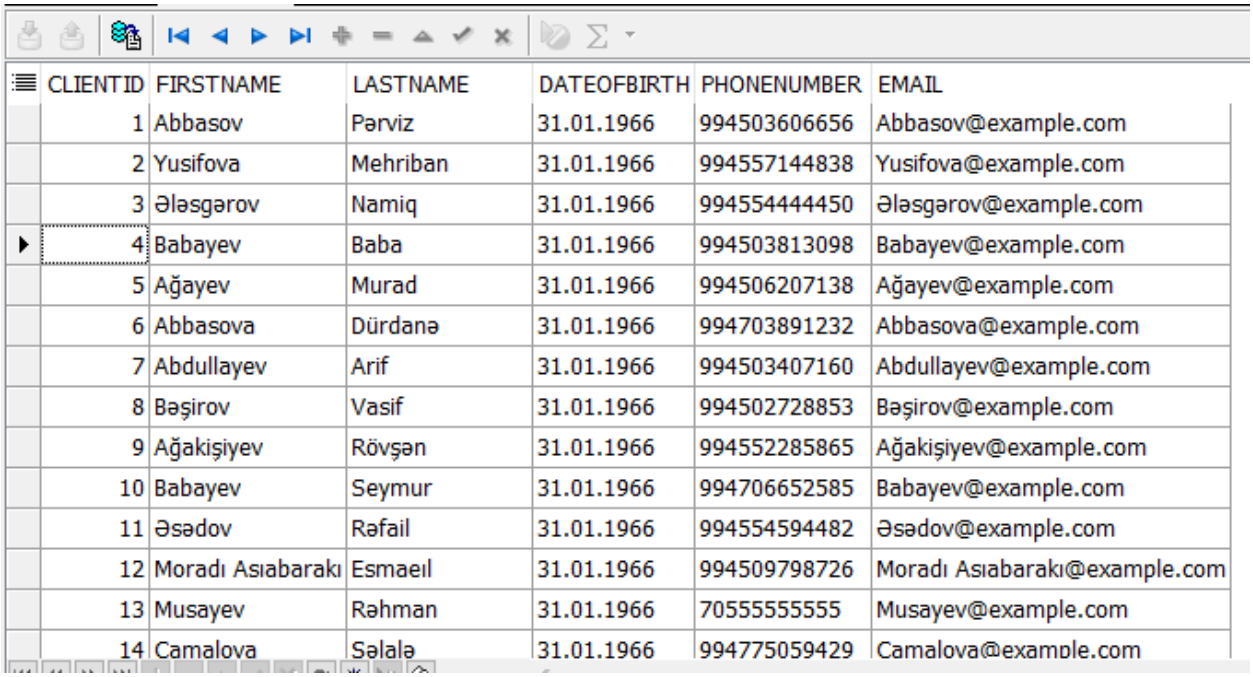
1. Мы выбираем столбцы "CLIENTID" (Идентификатор клиента), "FIRSTNAME" (Имя), "LASTNAME" (Фамилия), "DATEOFBIRTH" (Дата рождения), "PHONENUMBER" (Номер телефона) и "EMAIL" (Электронная почта) с помощью оператора SELECT.
2. Мы указываем таблицу "CLIENTS" в операторе FROM.
3. У нас есть условие фильтрации для каждого из заданных параметров: "v_mus_ad" (Имя клиента), "v_mus_soyad" (Фамилия клиента), "v_mus_number" (Номер телефона клиента) и "v_mus_mail" (Электронная почта клиента).
4. Мы используем операторы LIKE для выполнения частичного сопоставления строк и операторы IS NULL для обработки возможности пустых значений этих параметров.
5. В каждом условии фильтрации мы преобразуем значения столбцов и параметров в верхний регистр с помощью функции UPPER, чтобы выполнить поиск независимо от регистра символов.

Пример использования запроса:

```
SELECT
  CLIENTID, FIRSTNAME, LASTNAME, DATEOFBIRTH,
PHONENUMBER, EMAIL
```

```
FROM
   CLIENTS
WHERE
   1=1
   AND (UPPER(FIRSTNAME) LIKE '%' || UPPER(" || v_mus_ad || ") || '%' OR " || v_mus_ad || " IS NULL)
   AND (UPPER(LASTNAME) LIKE '%' || UPPER(" || v_mus_soyad || ") || '%' OR " || v_mus_soyad || " IS NULL)
   AND (PHONENUMBER LIKE '%' || " || v_mus_number || " || '%' OR " || v_mus_number || " IS NULL)
   AND (UPPER1f4e7_email LIKE '%' || UPPER(" || v_mus_mail || ") || '%' OR " || v_mus_mail || " IS NULL);
```

Этот запрос возвращает список клиентов банка, удовлетворяющих заданным фильтрам. Мы можем использовать этот запрос для получения требуемых данных о клиентах в нашей банковской среде. Значения параметров "v_mus_ad", "v_mus_soyad", "v_mus_number" и "v_mus_mail" можно задавать для поиска конкретных клиентов или оставлять пустыми для получения полного списка клиентов без фильтрации. Результат от этого запроса будет выглядеть следующим образом.



CLIENTID	FIRSTNAME	LASTNAME	DATEOFBIRTH	PHONENUMBER	EMAIL
1	Abbasov	Pərviz	31.01.1966	994503606656	Abbasov@example.com
2	Yusifova	Mehriban	31.01.1966	994557144838	Yusifova@example.com
3	Ələsgərov	Namiq	31.01.1966	994554444450	Ələsgərov@example.com
4	Babayev	Baba	31.01.1966	994503813098	Babayev@example.com
5	Ağayev	Murad	31.01.1966	994506207138	Ağayev@example.com
6	Abbasova	Dürdanə	31.01.1966	994703891232	Abbasova@example.com
7	Abdullayev	Arif	31.01.1966	994503407160	Abdullayev@example.com
8	Bəşirov	Vasif	31.01.1966	994502728853	Bəşirov@example.com
9	Ağakişiyev	Rövşən	31.01.1966	994552285865	Ağakişiyev@example.com
10	Babayev	Seymur	31.01.1966	994706652585	Babayev@example.com
11	Əsədov	Rəfail	31.01.1966	994554594482	Əsədov@example.com
12	Moradı Asıabarakı	Esmaeıl	31.01.1966	994509798726	Moradı Asıabarakı@example.com
13	Musayev	Rəhman	31.01.1966	70555555555	Musayev@example.com
14	Camalova	Səlalə	31.01.1966	994775059429	Camalova@example.com

3.2. Разработка пакета PL/SQL

3.2.1. Обзор PL/SQL

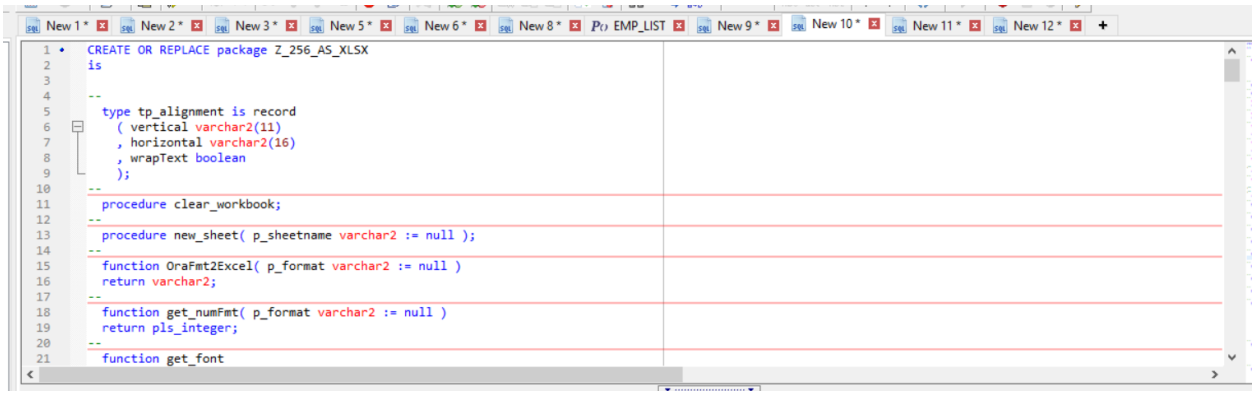
PL/SQL (Procedural Language/Structured Query Language) — это собственный процедурный язык программирования Oracle. Он расширяет язык SQL, добавляя процедурные конструкции, что позволяет разрабатывать сложные и многократно используемые приложения базы данных.

PL/SQL предоставляет надежную и безопасную среду для написания хранимых процедур, функций и пакетов. Он поддерживает управляющие структуры, циклы, обработку исключений и объявления переменных, что позволяет реализовать бизнес-логику и расширенные возможности обработки данных в базе данных.

3.2.2. Дизайн упаковки

Дизайн пакета PL/SQL вращается вокруг определения целей и функций, необходимых для создания отчетов Excel. Пакет выступает в качестве контейнера для связанных процедур, функций, переменных и курсоров, которые вместе выполняют нужные операции.

Spec-а пакета, где можно увидеть процедуры, типы данных и т.д.



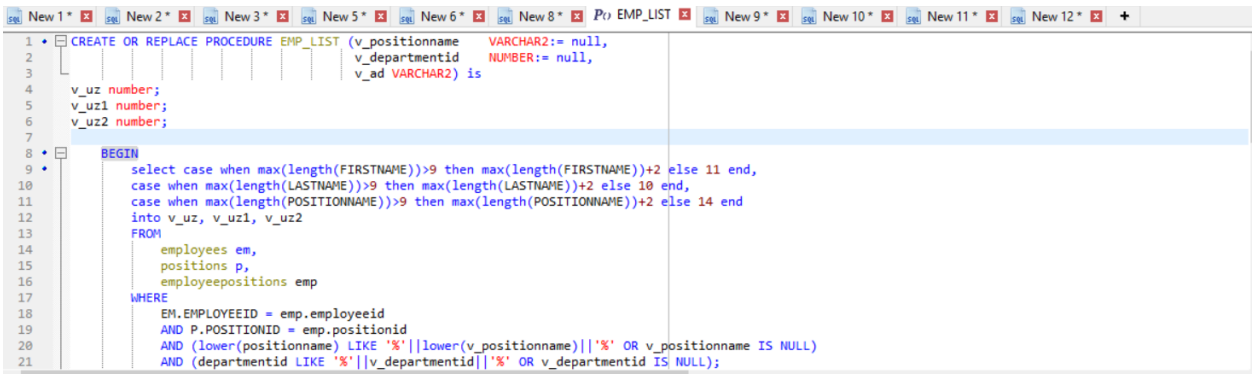
```
CREATE OR REPLACE package Z_256_AS_XLSX
is

--
  type tp_alignment is record
    ( vertical varchar2(11)
    , horizontal varchar2(16)
    , wrapText boolean
    );
--
  procedure clear_workbook;
--
  procedure new_sheet( p_sheetname varchar2 := null );
--
  function OraFmt2Excel( p_format varchar2 := null )
  return varchar2;
--
  function get_numFmt( p_format varchar2 := null )
  return pls_integer;
--
  function get_font
```

Дизайн пакета включает определение необходимых входных параметров, таких как критерии запроса или параметры отчета, а также указание выходного формата и структуры генерируемых отчетов Excel. Принимаются во внимание обработка ошибок, ведение журнала и любые дополнительные функции, необходимые для обработки или форматирования данных.

3.2.3. Реализация пакета

Реализация пакета PL/SQL включает в себя написание необходимого кода в теле пакета. Это включает в себя определение процедур и функций, которые выполняют определенные задачи, такие как выполнение запросов, извлечение данных и форматирование результатов.

Данная процедура EMP_LIST –возвращает список сотрудников банка в готовом Excel отчете по указанной директории.



```
CREATE OR REPLACE PROCEDURE EMP_LIST (v_positionname    VARCHAR2:= null,
                                      v_departmentid    NUMBER:= null,
                                      v_ad VARCHAR2) is
  v_uz number;
  v_uz1 number;
  v_uz2 number;

    BEGIN
        select case when max(length(FIRSTNAME))>9 then max(length(FIRSTNAME))+2 else 11 end,
        case when max(length(LASTNAME))>9 then max(length(LASTNAME))+2 else 10 end,
        case when max(length(POSITIONNAME))>9 then max(length(POSITIONNAME))+2 else 14 end
        into v_uz, v_uz1, v_uz2
        FROM
            employees em,
            positions p,
            employeepositions emp
        WHERE
            EM.EMPLOYEEID = emp.employeeid
            AND P.POSITIONID = emp.positionid
            AND (lower(positionname) LIKE '%'||lower(v_positionname)||'%' OR v_positionname IS NULL)
            AND (departmentid LIKE '%'||v_departmentid||'%' OR v_departmentid IS NULL);
```

Конструкции PL/SQL, такие как объявления курсора, присвоение переменных и условные операторы, используются для выполнения требуемых операций. Кроме того, пакет может использовать встроенные или пользовательские функции для преобразования или форматирования данных.

Опишем анализ кода пакета.

1. *Типы данных:*

- `tp_alignment` - пользовательский тип данных, определяющий выравнивание ячейки в Excel. Состоит из следующих полей:
  - `vertical` - вертикальное выравнивание (varchar2(11)).
  - `horizontal` - горизонтальное выравнивание (varchar2(16)).
  - `wrapText` - флаг переноса текста (boolean).

2. *Процедуры:*

- `clear_workbook` - очищает рабочую книгу (workbook) от данных.
- `new_sheet` - создает новый лист в рабочей книге. Принимает необязательный параметр `p_sheetname`, указывающий имя листа.
- `cell` - добавляет значение в указанную ячейку на листе. Поддерживает различные типы данных, такие как число, строка и дата.
- `hyperlink` - добавляет гиперссылку на указанную ячейку.
- `comment` - добавляет комментарий к указанной ячейке.
- `mergecells` - объединяет ячейки в прямоугольную область на листе.
- `list_validation` - добавляет правила проверки данных (validation) для диапазона ячеек.
- `defined_name` - определяет именованный диапазон ячеек на листе.
- `set_column_width` - устанавливает ширину столбца на листе.
- `set_column` - устанавливает форматирование для указанного столбца.
- `set_row` - устанавливает форматирование для указанной строки.

- `freeze_rows` - закрепляет указанное количество строк сверху при прокрутке листа.
- `freeze_cols` - закрепляет указанное количество столбцов слева при прокрутке листа.
- `freeze_pane` - закрепляет лист так, чтобы указанная ячейка стала верхней левой видимой ячейкой.
- `set_autofilter` - применяет автофильтр к указанной области на листе.
- `save` - сохраняет рабочую книгу в формате XLSX по указанному пути и с указанным именем файла.
- `query2sheet` - выполняет SQL-запрос и добавляет результаты на лист.

2. *Функции:*
- `create_workbook` - создает новую рабочую книгу Excel. Принимает необязательный параметр `p_filename`, указывающий имя файла.
- `get_sheet_names` - возвращает список имен листов в рабочей книге.
- `get_row_count` - возвращает количество строк на указанном листе.
- `get_column_count` - возвращает количество столбцов на указанном листе.
- `get_cell_value` - возвращает значение указанной ячейки на листе.
- `get_column_width` - возвращает ширину указанного столбца на листе.
- `get_column_format` - возвращает форматирование указанного столбца на листе.
- `get_row_format` - возвращает форматирование указанной строки на листе.
- `set_sheet_name` - изменяет имя указанного листа в рабочей книге.
- `set_cell_format` - устанавливает форматирование для указанной ячейки на листе.
- `set_sheet_visibility` - устанавливает видимость указанного листа в рабочей книге.
- `delete_sheet` - удаляет указанный лист из рабочей книги.
- `get_sheet_index` - возвращает индекс указанного листа в рабочей книге.

- `move_sheet` - перемещает указанный лист на указанную позицию в рабочей книге.
- `copy_sheet` - создает копию указанного листа в рабочей книге.
- `get_hyperlink` - возвращает гиперссылку, связанную с указанной ячейкой на листе.
- `get_comment` - возвращает комментарий, связанный с указанной ячейкой на листе.
- `get_defined_names` - возвращает список именованных диапазонов на указанном листе.
- `get_autofilter_range` - возвращает область, на которую применен автофильтр на указанном листе.
- `remove_autofilter` - удаляет автофильтр с указанного листа.
- `add_image` - добавляет изображение на указанный лист.

Это основные функции, предоставляемые в пакете ` Z_256_AS_XLSX`. Они позволяют создавать, форматировать, заполнять данными и управлять Excel-отчетами в базе данных Oracle.

3.2.4. Создание отчета Excel

Пакет PL/SQL, разработанный для создания отчетов Excel, использует различные механизмы для запросов к базе данных, извлечения соответствующих данных и их форматирования таким образом, чтобы их можно было экспортировать в формат Excel.

Пакет может включать процедуры, которые принимают такие параметры, как диапазоны дат, условия фильтрации или определенные критерии для извлечения данных. Эти параметры используются для построения динамических SQL-запросов, что обеспечивает гибкость при создании отчетов на основе пользовательских требований.

После извлечения данных их можно обработать в пакете PL/SQL, чтобы отформатировать их соответствующим образом для экспорта в Excel. Это может включать в себя применение стилей, настройку ширины столбцов, объединение ячеек или добавление верхних и нижних колонтитулов. PL/SQL предоставляет встроенные функции для манипулирования строками и обработки форматов даты и числа, обеспечивая точное представление данных в отчетах Excel.

Пример параметров:

```
workbook.sheets( t_sheet ).row_fmts( p_row ).numFmtId := p_numFmtId;
workbook.sheets( t_sheet ).row_fmts( p_row ).fontId := p_fontId;
workbook.sheets( t_sheet ).row_fmts( p_row ).fillId := p_fillId;
workbook.sheets( t_sheet ).row_fmts( p_row ).borderId := p_borderId;
workbook.sheets( t_sheet ).row_fmts( p_row ).alignment := p_alignment;
```

Для облегчения экспорта в Excel пакет может использовать функциональные возможности Oracle для создания файлов XLSX (значения, разделенные запятыми). Отформатировав данные, полученные из базы данных, в формате XLSX, их можно легко импортировать в Excel или преобразовать в формат Excel с помощью внешних инструментов или библиотек.

Кроме того, пакет может использовать внешние библиотеки или API-интерфейсы, специально разработанные для работы с Excel, для прямого создания файлов Excel. Эти библиотеки обеспечивают расширенный контроль над форматированием. параметры, стили ячеек и возможность создавать несколько рабочих листов или применять формулы в отчетах Excel.

3.2.5. Создание процедуры с использованием пакета и внедрением нашего запроса в эту процедуру.

```
CREATE OR REPLACE PROCEDURE EMP_LIST (v_positionname    VARCHAR2,
                                      v_departmentid    NUMBER,
                                      v_ad VARCHAR2) is
v_uz number;
v_uz1 number;
v_uz2 number;

    BEGIN
        select case when max(length(FIRSTNAME))>9 then max(length(FIRSTNAME))+2 else 11 end,
        case when max(length(LASTNAME))>9 then max(length(LASTNAME))+2 else 10 end,
        case when max(length(POSITIONNAME))>9 then max(length(POSITIONNAME))+2 else 14 end
        into v_uz, v_uz1, v_uz2
        FROM
            employees em,
            positions p,
            employeepositions emp
        WHERE
            EM.EMPLOYEEID = emp.employeeid
            AND P.POSITIONID = emp.positionid
            AND (lower(positionname) LIKE '%'||lower(v_positionname)||'%' OR v_positionname IS NULL)
            AND (departmentid LIKE '%'||v_departmentid||'%' OR v_departmentid IS NULL);

        z_256_as_xlsx.CLEAR_WORKBOOK;
        z_256_as_xlsx.new_sheet;
        z_256_as_xlsx.set_column_width(1, 12, 1);
        z_256_as_xlsx.set_column_width(2, v_uz, 1);
        z_256_as_xlsx.set_column_width(3, v_uz1, 1);
        z_256_as_xlsx.set_column_width(4, 13, 1);
        z_256_as_xlsx.set_column_width(5, 14, 1);
        z_256_as_xlsx.set_column_width(6, 12, 1);
        z_256_as_xlsx.set_column_width(7, v_uz2, 1);
        z_256_as_xlsx.set_row(2,p_borderId => z_256_as_xlsx.get_border( 'medium ', 'medium', 'medium', 'medium' ));
        z_256_as_xlsx.set_row(3,p_borderId => z_256_as_xlsx.get_border( 'medium ', 'medium', 'medium', 'medium' ));
        z_256_as_xlsx.set_row(4,p_borderId => z_256_as_xlsx.get_border( 'medium ', 'medium', 'medium', 'medium' ));
        z_256_as_xlsx.set_row(5,p_borderId => z_256_as_xlsx.get_border( 'medium ', 'medium', 'medium', 'medium' ));
        z_256_as_xlsx.QUERY2SHEET('SELECT
        em.*,
        p.*
    FROM
        employees em,
        positions p,
        employeepositions emp
    WHERE
        EM.EMPLOYEEID = emp.employeeid
        AND P.POSITIONID = emp.positionid
        AND (lower(positionname) LIKE ''%''||lower(''''||v_positionname||'''')||''%'' OR ''''||v_positionname||''''' IS NULL)
        AND (departmentid LIKE ''%''||''||v_departmentid||''||''%'' OR '||v_departmentid||'IS NULL)'
        ,1
        ,p_fillId => z_256_as_xlsx.get_fill( 'solid', 'E2EFDA' )
        ,p_borderId => z_256_as_xlsx.get_border( 'medium ', 'medium', 'medium', 'medium' ));
        z_256_as_xlsx.SAVE('YUNSUR', ''||v_ad||'.Xlsx');
    END;
```

Объяснение процедуры:

1. Процедура `EMP_LIST` принимает три входных параметра: `v_positionname` (название должности), `v_departmentid` (идентификатор отдела) и `v_ad` (имя файла отчета).

2. Внутри процедуры определены три переменные `v_uz`, `v_uz1` и `v_uz2`, которые будут использоваться для вычисления ширины столбцов в отчете.

3. Затем выполняется запрос, который находит максимальную длину полей `FIRSTNAME`, `LASTNAME` и `POSITIONNAME` из таблиц `employees`, `positions` и `employeepositions` соответственно. Полученные значения сохраняются в переменных `v_uz`, `v_uz1` и `v_uz2`.

4. Далее происходит очистка рабочей книги с помощью вызова функции `z_256_as_xlsx.CLEAR_WORKBOOK`.

5. Создается новый лист с помощью вызова функции `z_256_as_xlsx.new_sheet`.

6. Устанавливается ширина столбцов с помощью вызовов функции `z_256_as_xlsx.set_column_width`. Указывается номер первого и последнего столбца, а также ширина столбца.

7. Устанавливается форматирование строк с помощью вызовов функции `z_256_as_xlsx.set_row`. Указывается номер строки и идентификаторы форматирования (здесь используется функция `z_256_as_xlsx.get_border` для получения идентификатора границы).

8. Заполняется лист данными из запроса с помощью вызова функции `z_256_as_xlsx.QUERY2SHEET`. Первый аргумент - SQL-запрос, второй аргумент - номер строки, с которой начинается заполнение, и последующие аргументы - идентификаторы форматирования (здесь используется функция `z_256_as_xlsx.get_fill` для получения идентификатора заливки и `z_256_as_xlsx.get_border` для получения идентификатора границы).

9. Сохраняется отчет с помощью вызова функции `z_256_as_xlsx.SAVE`. Указывается имя файла и путь сохранения (здесь используется имя файла `v_ad`).

Создадим еще одну процедуру для другого нашего запроса, который возвращает нам список клиентов банка.

```
CREATE OR REPLACE PROCEDURE CLI_LIST (v_mus_ad      VARCHAR2,
                                      v_mus_soyad   VARCHAR2,
                                      v_mus_number  VARCHAR2,
                                      v_mus_mail    VARCHAR2,
                                      v_ad          VARCHAR2) is
v_uz number;
v_uz1 number;
v_uz3 number;

    BEGIN
        select case when max(length(FIRSTNAME))>9 then max(length(FIRSTNAME))+2 else 11 end,
        case when max(length(LASTNAME))>9 then max(length(LASTNAME))+2 else 10 end,
        case when max(length(EMAIL))>9 then max(length(EMAIL))+2 else 14 end
        into v_uz, v_uz1, v_uz3
        FROM
            CLIENTS
        WHERE
            1=1
            AND (UPPER(FIRSTNAME) LIKE '%'||UPPER(v_mus_ad)||'%' OR v_mus_ad IS NULL)
            AND (UPPER(LASTNAME) LIKE '%'||UPPER(v_mus_soyad)||'%' OR v_mus_soyad IS NULL)
            AND (PHONENUMBER LIKE '%'||v_mus_number||'%' OR v_mus_number IS NULL)
            AND (UPPER(EMAIL) LIKE '%'||UPPER(v_mus_mail)||'%' OR v_mus_mail IS NULL);


      z_256_as_xlsx.CLEAR_WORKBOOK;
      z_256_as_xlsx.new_sheet;
      z_256_as_xlsx.set_column_width(1, 12, 1);
      z_256_as_xlsx.set_column_width(2, v_uz, 1);
      z_256_as_xlsx.set_column_width(3, v_uz1, 1);
      z_256_as_xlsx.set_column_width(4, 15, 1);
      z_256_as_xlsx.set_column_width(5, 15, 1);
      z_256_as_xlsx.set_column_width(6, v_uz3, 1);
      z_256_as_xlsx.QUERY2SHEET('SELECT
                                    CLIENTID, FIRSTNAME, LASTNAME, DATEOFBIRTH, PHONENUMBER, EMAIL
                                FROM
                                    CLIENTS
                                WHERE
                                    1=1
                                    AND (UPPER(FIRSTNAME) LIKE ''%''||UPPER('''||v_mus_ad||''')||''%'' OR '''||v_mus_ad||''' IS NULL)
                                    AND (UPPER(LASTNAME) LIKE ''%''||UPPER('''||v_mus_soyad||''')||''%'' OR '''||v_mus_soyad||''' IS NULL)
                                    AND (PHONENUMBER LIKE ''%''||'''||v_mus_number||'''||''%'' OR '''||v_mus_number||''' IS NULL)
                                    AND (UPPER(EMAIL) LIKE ''%''||UPPER('''||v_mus_mail||''')||''%'' OR '''||v_mus_mail||''' IS NULL)'
      ,1
      ,p_fillId => z_256_as_xlsx.get_fill( 'solid', 'E2EFDA' )
      ,p_borderId => z_256_as_xlsx.get_border( 'medium ', 'medium', 'medium', 'medium' ));
      z_256_as_xlsx.SAVE('YUNSUR', v_ad||'.Xlsx');
  END;
```

Этот блок выполняет запрос к таблице `CLIENTS` для вычисления максимальной длины полей `FIRSTNAME`, `LASTNAME` и `EMAIL`. Результаты запроса сохраняются в переменные `v_uz`, `v_uz1` и `v_uz3`.

sql

-- Очистка рабочей книги

z_256_as_xlsx.CLEAR_WORKBOOK;

-- Создание нового листа

z_256_as_xlsx.new_sheet;

```
-- Установка ширины столбцов
z_256_as_xlsx.set_column_width(1, 12, 1);
z_256_as_xlsx.set_column_width(2, v_uz, 1);
z_256_as_xlsx.set_column_width(3, v_uz1, 1);
z_256_as_xlsx.set_column_width(4, 15, 1);
z_256_as_xlsx.set_column_width(5, 15, 1);
z_256_as_xlsx.set_column_width(6, v_uz3, 1);
```

В этом блоке происходит подготовка рабочей книги перед заполнением данными. Сначала вызывается функция `CLEAR_WORKBOOK` из пакета `z_256_as_xlsx`, чтобы

очистить содержимое рабочей книги. Затем вызывается функция `new_sheet`, чтобы создать новый лист в рабочей книге. Далее вызываются функции `set_column_width` для установки ширины столбцов в Excel-листе. Ширина столбцов зависит от значений переменных `v_uz`, `v_uz1` и `v_uz3`, которые были определены ранее.

sql

```
-- Заполнение листа данными
z_256_as_xlsx.QUERY2SHEET(
  'SELECT
    CLIENTID, FIRSTNAME, LASTNAME, DATEOFBIRTH, PHONENUMBER, EMAIL
  FROM
    CLIENTS
  WHERE
    1 = 1
```

```
        AND (UPPER(FIRSTNAME) LIKE "%" || UPPER('" || v_mus_ad || "') || "%" OR '" || v_mus_ad || "' IS NULL)
        AND (UPPER(LASTNAME) LIKE "%" || UPPER('" || v_mus_soyad || "') || "%" OR '" || v_mus_soyad || "' IS NULL)
        AND (PHONENUMBER LIKE "%" || '" || v_mus_number || "' || "%" OR '" || v_mus_number || "' IS NULL)
        AND (UPPER(EMAIL) LIKE "%" || UPPER('" || v_mus_mail || "') || "%" OR '" || v_mus_mail || "' IS NULL)',
    1,
    p_fillId => z_256_as_xlsx.get_fill('solid', 'E2EFDA'),
    p_borderId => z_256_as_xlsx.get_border('medium ', 'medium', 'medium', 'medium')
  );
```

Этот блок использует функцию `QUERY2SHEET` из пакета `z_256_as_xlsx`, чтобы заполнить лист данными из таблицы `CLIENTS`. Запрос включает фильтры на основе переданных параметров `v_mus_ad`, `v_mus_soyad`, `v_mus_number` и `v_mus_mail`. Результаты запроса помещаются на лист в Excel-файле. Кроме того, используются функции `get_fill` и `get_border` для задания стилей заполнения и границ ячеек в Excel.

```
sql
  -- Сохранение рабочей книги в файл
  z_256_as_xlsx.SAVE('YUNSUR', v_ad || '.Xlsx');
END;
```

В конце процедуры вызывается функция `SAVE` из пакета `z_256_as_xlsx`, чтобы сохранить рабочую книгу в файл с именем, указанным в параметре `v_ad`, с расширением `.Xlsx`.

Надеюсь, это более подробное объяснение поможет вам лучше понять структуру и функциональность процедуры `CLI_LIST`.

3.3. Тестирование и проверка

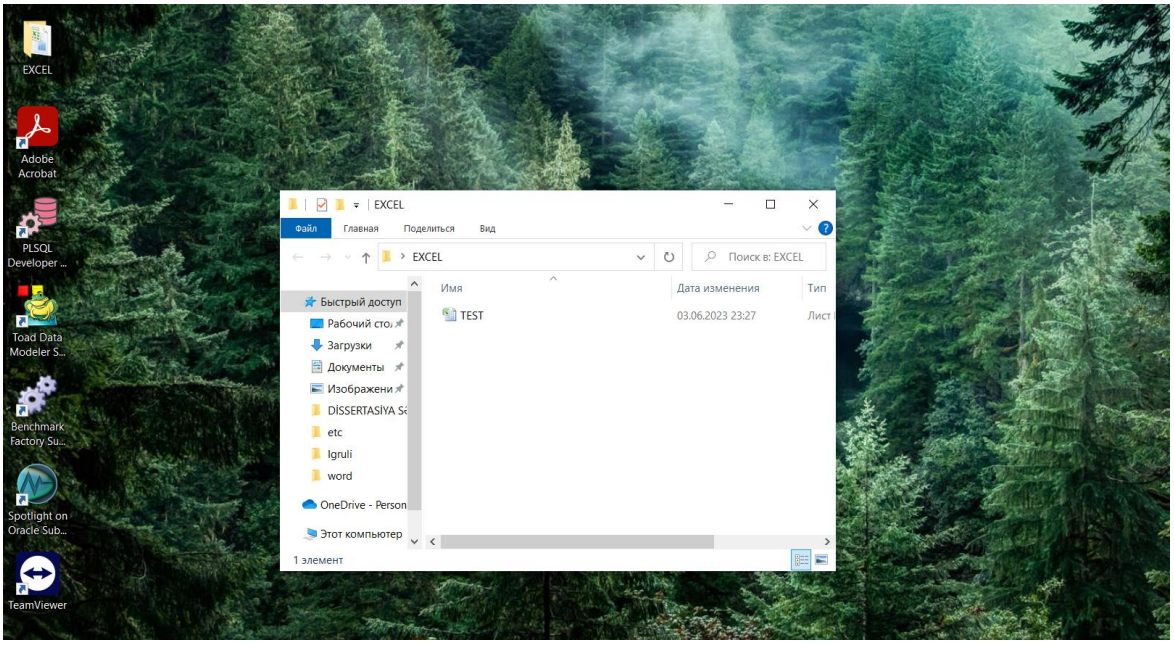
Чтобы гарантировать точность и надежность реализованного решения, выполняются строгие процедуры тестирования и проверки. Это включает в себя проверку функциональности созданной базы данных, правильности структуры таблиц, точности заполнения данных и успешного создания отчетов Excel.

EXEC EMP_LIST (NULL, NULL, 'TEST');

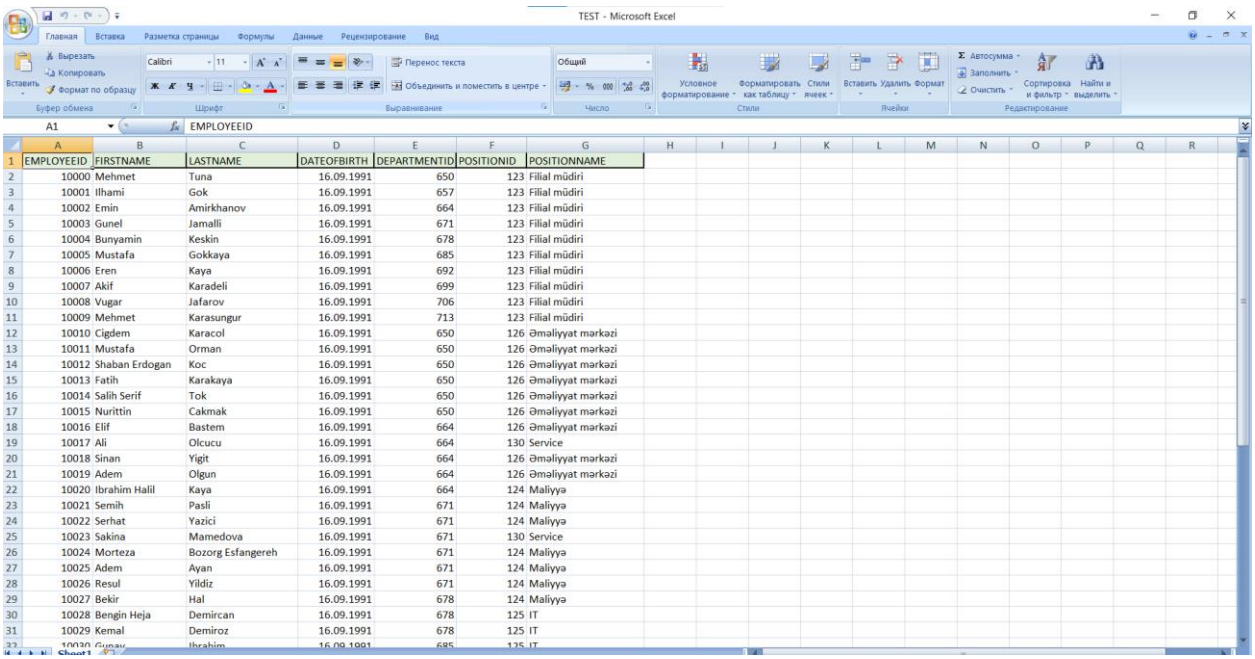
Если запустить этот скрипт, то автоматически создаться Excel отчет в папке EXCEL, который находится на рабочем столе (в нашем случае). Он вернет полный список сотрудников банка, так как в параметрах процедуры указаны значения NULL, а по логике нашего кода это возвращает все возможные данные из таблицы. Но можно вести 2 разных условия для создания таблицы такие как: Позиция, Идентификатор департамента.

Результат запуска скрпита:

Создался Excel под названием TEST который мы указывали в процедуре как третий параметр.



И как можем заметить Excel заполнен соответствующей информацией. Это список сотрудников банка.



Различные методы тестирования, такие как модульное тестирование, интеграционное тестирование и системное тестирование, используются для оценки различных аспектов реализации. Сценарии тестирования и тестовые данные определены таким образом, чтобы охватить широкий диапазон возможных входных данных и сценариев для обеспечения комплексного тестирования.

Валидация реализованного решения включает в себя сравнение сгенерированных отчетов с ожидаемыми результатами и проверку целостности данных. Это делается для подтверждения того, что отчеты точно представляют нужную информацию и что данные, извлеченные из базы данных, непротиворечивы и надежны.

Тестирование и проверка также включают в себя тестирование производительности для оценки эффективности и скорости отклика операций базы данных и процесса создания отчетов. Это включает в себя анализ времени отклика, использования ресурсов и масштабируемости реализованного решения при различных условиях нагрузки.

3.4. Заключение

В заключение в этой главе представлен всесторонний обзор практической реализации диссертации. Процесс включает в себя создание базы данных Oracle, включая проектирование таблиц и заполнение данными. Кроме того, обсуждалась разработка пакета PL/SQL для создания отчетов Excel.

Следуя описанным шагам, от создания базы данных до генерации отчетов, реализованное решение демонстрирует успешное применение теоретических концепций в реальном контексте. Сочетание мощных функций Oracle и гибкости PL/SQL позволяет создать надежную и эффективную систему баз данных, позволяющую получать ценную информацию с помощью исчерпывающих отчетов Excel.

Процедуры тестирования и проверки обеспечивают точность и надежность реализованного решения, обеспечивая уверенность в практических выводах диссертации. В целом практическая реализация служит прочной основой для последующих глав и способствует общему успеху исследовательского проекта.

3.5. Ограничения и будущие улучшения

Важно признать ограничения реализованного решения и определить области для будущих улучшений. Некоторые потенциальные ограничения и соображения включают:

- Масштабируемость. По мере роста базы данных и размера данных производительность запросов и создания отчетов может снижаться. Реализация таких методов, как секционирование, индексирование или стратегии оптимизации запросов, может решить проблемы масштабируемости.

- Безопасность: обеспечение безопасности данных и контроля доступа в базе данных имеет решающее значение. Дополнительные меры безопасности, такие как шифрование, аутентификация пользователей и механизмы авторизации, должны быть реализованы для защиты конфиденциальных данных.

- Обработка ошибок. В пакет PL/SQL должны быть включены надежные механизмы обработки ошибок, чтобы корректно справляться с непредвиденными ситуациями, такими как несогласованность данных, неверные входные данные или проблемы с подключением к базе данных.

- Гибкость отчетности: можно внести улучшения, чтобы обеспечить большую гибкость при создании отчетов. Это может включать в себя включение удобных интерфейсов для указания параметров отчета, возможность создания специальных запросов или поддержку настраиваемых макетов отчетов.

- Интеграция с внешними инструментами. Интеграция пакета PL/SQL с внешними инструментами отчетности или визуализации может предоставить более продвинутые возможности отчетности, интерактивные информационные панели и расширенные возможности визуализации данных.

- Автоматизация и планирование. Добавление функций автоматизации в пакет PL/SQL, таких как создание отчетов по расписанию или доставка по

электронной почте, может повысить эффективность и удобство использования решения.

- Оптимизация производительности: Непрерывный мониторинг и оптимизация производительности запросов к базе данных и процесса генерации отчетов имеют важное значение. Такие методы, как настройка запросов, оптимизация индексов и механизмы кэширования, могут повысить общую производительность системы.

- Отзывы пользователей и итеративные улучшения: сбор отзывов от конечных пользователей и заинтересованных сторон может предоставить ценную информацию для итеративных улучшений. Включение отзывов пользователей в решение гарантирует, что оно будет соответствовать меняющимся потребностям и требованиям пользователей.

РЕЗУЛЬТАТЫ

В этой работе были освещены практические аспекты диссертации с упором на создание базы данных Oracle, создание таблиц, заполнение данными. и разработка пакета PL/SQL для создания отчетов Excel. Были обсуждены подробные шаги и соображения, связанные с каждым этапом процесса внедрения.

Используя возможности Oracle и гибкость PL/SQL, была создана надежная и эффективная система баз данных. Возможность создавать отчеты Excel на основе пользовательских запросов повышает удобство использования и ценность базы данных, позволяя пользователям извлекать важные сведения из данных.

Хотя реализованное решение демонстрирует успешную практическую реализацию диссертации, есть области для будущих улучшений и соображений по устранению ограничений. Включая меры масштабируемости, безопасности, гибкости и оптимизации производительности, решение может быть дополнительно улучшено для удовлетворения меняющихся потребностей и обеспечения его долгосрочной эффективности.

В целом практическая реализация подтверждает результаты исследования и служит реальной демонстрацией теоретических концепций, представленных в предыдущих главах. Информация, полученная из базы данных и сгенерированных отчетов Excel, способствует общему успеху диссертации и обеспечивает ценную основу для будущих исследований и приложений в этой области.

ЛИТЕРАТУРА

1. Джеллер Дж., Джеллер П. SQL: руководство по изучению языка. - СПб.: Питер, 2019.
2. Бекман К., Харрис Д. SQL All-in-One For Dummies. - Wiley, 2018.
3. Грофф Дж., Вайнберг П., Уэйнер П. SQL: Структурированный язык запросов Молина М. SQL Cookbook. - O'Reilly Media, 2015.
4. Кафка Б., Рабинович С. Oracle PL/SQL for DBAs: Security, Scheduling, Performance & More. - Rampant TechPress, 2017.
5. МакДональд М. Excel 2019 Power Programming with VBA. - Wiley, 2019.
6. Каллахан М., Джиоттоли Г. Майкрософт Excel для финансистов. - Вильямс, 2018.
7. Официальная документация Oracle для SQL и PL/SQL.