МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Реализация базы данных банка с применением средств мониторинга состояния СУБД»

Выполнил студент Макаров А.И.

(Ф.И.О.)

Руководитель работы асс. Нистюк О.А.

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

И.о. зав. кафедрой ст. преп. Блинова Е.А.

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

Курсовая работа защищена с оценкой

Минск 2023

Содержание

[Введение 6](#_Toc154057679)

[1 Постановка задачи 7](#_Toc154057680)

[2 Проектирование базы данных 8](#_Toc154057681)

[2.1 Схема базы данных 8](#_Toc154057682)

[2.2 Таблицы базы данных 10](#_Toc154057683)

[2.3 Вывод по разделу 12](#_Toc154057684)

[3 Разработка объектов базы данных 13](#_Toc154057685)

[3.1 Табличное пространство 13](#_Toc154057686)

[3.1 Таблицы 13](#_Toc154057687)

[3.2.1 Таблицы 13](#_Toc154057688)

[3.3 Процедуры 14](#_Toc154057689)

[3.4 Функции 15](#_Toc154057690)

[3.5 Триггеры базы данных 15](#_Toc154057691)

[3.6 Вывод по разделу 16](#_Toc154057692)

[4 Описание процедур экспорта и импорта 17](#_Toc154057693)

[4.1 Описание процедуры экспорта 17](#_Toc154057694)

[4.2 Описание процедуры импорта 18](#_Toc154057695)

[4.3 Вывод по разделу 19](#_Toc154057696)

[5 Тестирование производительности 19](#_Toc154057697)

[5.1 Оптимизирование запросов 19](#_Toc154057698)

[5.2 Проверка выполнения хранимых процедур 21](#_Toc154057699)

[5.2 Вывод по разделу 23](#_Toc154057700)

[6 Описание и применение технологии в БД 24](#_Toc154057701)

[6.1 Описание взаимодействия с технологией 24](#_Toc154057702)

[6.2 Вывод по разделу 26](#_Toc154057703)

[7 Руководство пользователя 28](#_Toc154057704)

[Заключение 29](#_Toc154057705)

[Список используемых источников 30](#_Toc154057706)

[Приложение A Создание таблиц 31](#_Toc154057707)

[Приложение Б Процедуры и функции 33](#_Toc154057708)

[Приложение С Создание триггера 54](#_Toc154057709)

# **Введение**

Современная банковская система требует эффективного и безопасного управления данными для обеспечения стабильности, надежности и оперативности финансовых операций. Реализация базы данных для банка становится ключевым аспектом в обеспечении высококачественного обслуживания клиентов, а также в поддержании прозрачности и эффективности внутренних банковских процессов.

Цель данного курсового проекта заключается в исследовании и реализации базы данных банка с применением средств мониторинга состояния системы управления базами данных (СУБД). Мониторинг состояния базы данных играет важную роль в обеспечении стабильности и производительности системы, а также в оперативном выявлении и устранении возможных проблем.

В данном проекте будет рассмотрено использование современных технологий в области управления данными, таких как системы управления базами данных, а также инструменты мониторинга, спроектированные для отслеживания и анализа работы баз данных.

Курсовой проект предполагает подробное изучение проблем и задач, связанных с реализацией базы данных в банковской среде, а также выявление средств мониторинга, способных обеспечить высокий уровень доступности и производительности системы.

Top of Form

Bottom of Form

# **1 Постановка задачи**

В рамках данного проекта требуется спроектировать инфраструктуру базы данных для банка. Для этого необходимо провести анализ требований и определить следующие элементы и их содержимое в нашей базе данных: таблицы и связи между ними, ограничения целостности, профили безопасности, пользователи, триггеры, хранимые процедуры, функции и индексы. Затем необходимо разработать эти объекты в базе данных, используя СУБД Oracle 19c и написание SQL-скриптов.

Для заполнения таблиц данными предусмотрен импорт из JSON-файлов. Кроме того, будет использована технология Oracle Enterprise Manager для анализа состояния базы данных в определенном временном интервале.

Важными функциями приложения будут:

* определение ролей (главный администратор, администратор, пользователь);
* управление клиентами (добавление, удаление, изменение);
* управление счетами (добавление, удаление, изменение);
* анализ данных (количество операций по счетам по периодам, оборот по всем счетам, общее число счетов);
* обеспечение ограничений по остатку на счете.
* История по счету
* Проведение операций по счетам (депозиты, снятия, переводы между счетами, пополнение)

Реализация этих операций будет осуществляться с помощью хранимых процедур и функций.

Важным этапом проекта будет тестирование производительности базы данных на таблице, содержащей не менее 100 000 строк. В случае необходимости будут внесены изменения в структуру базы данных для оптимизации производительности.

Применить технологию Oracle Enterprise Manager необходимо для анализа состояния базы данных, затрачиваемых и доступных ресурсов, объем доступных ресурсов.

2 Проектирование базы данных

2.1 Схема базы данных

Для начала создадим отдельную базу данных для курсового проекта с помощью утилиты Database Configuration Assistant. Далее определим, какие таблицы нам понадобятся для хранения информации о пользователях и их счетах.

В базе банка, может быть множество таблиц, в зависимости от требований и функциональности проекта. Давайте ограничимся основными таблицами, необходимыми для хранения информации о пользователях, счетах и операциях. Предварительная диаграмма таблиц и их отношения представлены на рисунке 2.1.

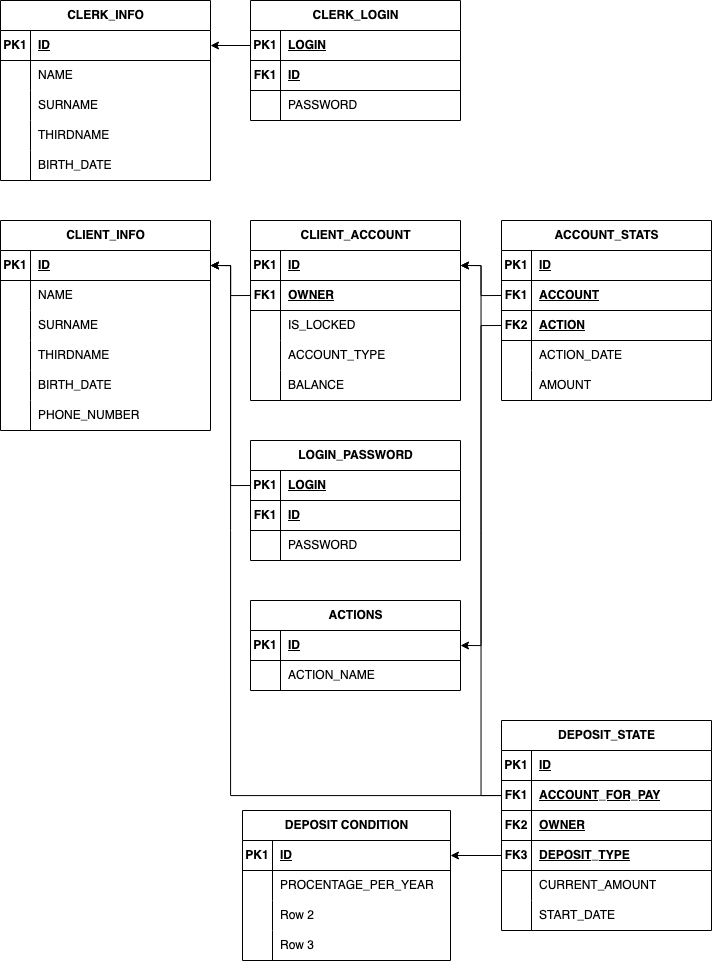


Рисунок 2.1 – Схема базы данных

Также в этих таблицах указаны предварительные типы данных для столбцов, которые могут измениться при необходимости.

Для эффективного использования системы важно определить роли пользователей и их сценарии использования. Сценарии описывают, как каждый пользователь будет взаимодействовать с системой в соответствии с их ролью. Это помогает определить доступные функции, доступные данные и организацию навигации в системе. Для визуализации взаимодействия между пользователями и системой используются диаграммы UML.

Роли пользователей определяют набор прав и привилегий, предоставляемых пользователям в системе. Каждая роль имеет свои уникальные возможности и доступ к функциям системы. В вашем проекте роли пользователей включают Admin (администратор) и User (пользователь). Это означает, что администратор будет иметь расширенные возможности и полный доступ ко всем функциям системы, в то время как пользователь будет иметь ограниченный доступ только к определенным функциям и данным, соответствующим его роли.

Далее определим, как будет происходить взаимодействие с системой. Наглядно это продемонстрирует Use-case диаграмма – рисунок 2.2.

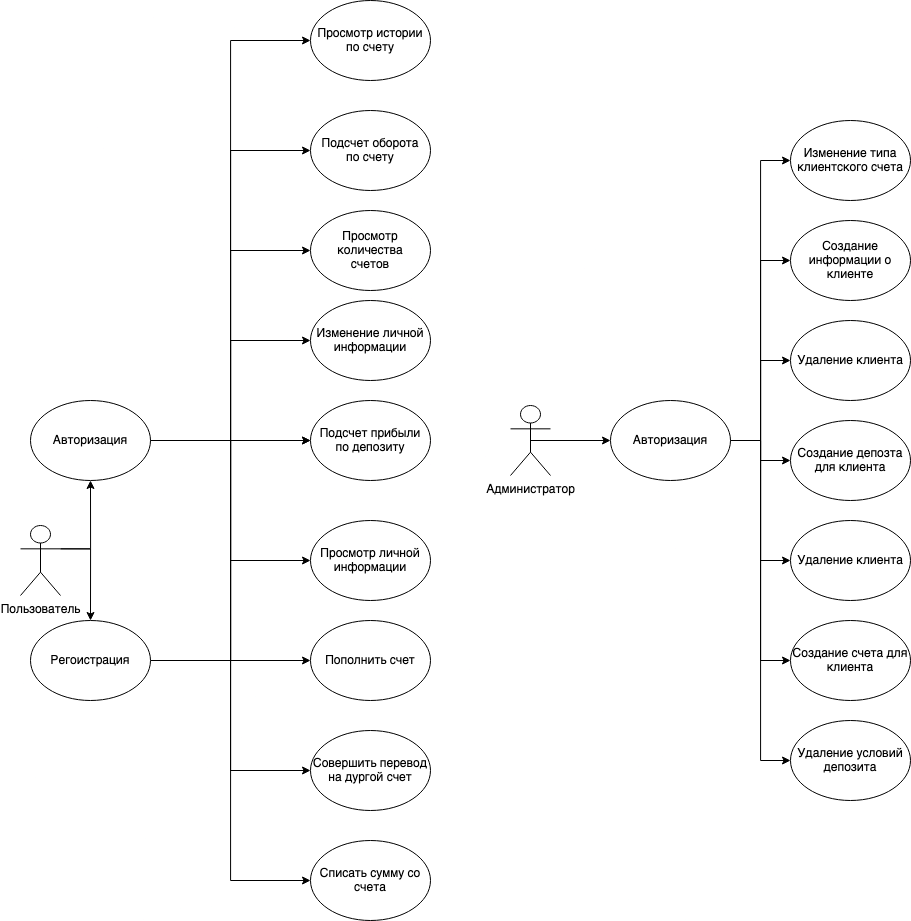


Рисунок 2.2 – Use Case диаграмма

Как мы установили ранее, нам понадобится две группы пользователей – Администраторы (пользователи, которые выполняют управление системой), а также обычные пользователи. Все действия данных пользователей будут выполняться через хранимые процедуры и функции.

2.2 Таблицы базы данных

Теперь опишем таблицы базы данных: названия, названия столбцов, типы данных столбцов, описание содержания столбцов. Описание представлено в таблицах 2.1 – 2.9.

Таблица 2.1 – Содержание таблицы CLERK\_INFO(Профиль работника банка)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | NUMBER | Уникальный идентификатор пользователя |
| NAME | Nvarchar2(50) | Имя пользователья |
| SURNAME | Nvarchar2(50) | Фамилия пользователя |
| THIRDNAME | Nvarchar2(50) | Отчество пользователя |
| BIRTH\_DATE | date | Дата рождения пользователя |

Эта таблица хранит в себе данные о пользовательских профилях.

Таблица 2.2 – Содержание таблицы CLERK\_LOGIN\_PASSWORD (Данные для авторизации работников банка)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | INT | Уникальный идентификатор пользователя |
| LOGIN | Nvarchar2(30) | Логин пользователя |
| PASSWORD | Nvarchar2(30) | Пароль пользователя |

В этой таблице содержатся данные для авторизации работников банка

Таблица 2.3 – Содержание таблицы LOGIN\_PASSWORD (Данные для авторизации клиентов банка)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | INT | Уникальный идентификатор пользователя |
| LOGIN | Nvarchar2(30) | Логин пользователя |
| PASSWORD | Nvarchar2(30) | Пароль пользователя |

В этой таблице содержатся данные для авторизации клиентов банка

Таблица 2.4 – Содержание таблицы CLIENT\_INFO (Личные данные клиентов банка)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | NUMBER | Уникальный идентификатор пользователя |
| NAME | Nvarchar2(50) | Имя пользователья |
| SURNAME | Nvarchar2(50) | Фамилия пользователя |
| THIRDNAME | Nvarchar2(50) | Отчество пользователя |
| BIRTH\_DATE | date | Дата рождения пользователя |
| PHONE\_NUMBER | VARCHAR(30) | Номер телефона клиента |

В данной таблице хранятся личные данные клиентов банка

Таблица 2.5 – Содержание таблицы CLIENT\_ACCOUNT (Данные всех счетов пользователей банка)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | INT | Уникальный идентификатор счета |
| OWNER | INT | Идентификатор пользователя, которому принадлежит счет |
| IS\_LOCKED | INT | Указание того, заблокирован счет или нет |
| ACCOUNT\_TYPE | INT | Функция получения ID жанра, возвращает INT. |
| BLANCE | FLOAT | Сумма на счете в данный момент |

В данной таблице содержатся данные необходимые для проведения операций по счетам клиентов банка

Таблица 2.6 – Содержание таблицы ACCOUNT\_STATS (История операций по всем счетам пользователя)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ACCOUNT | INT | Идентификатор счета с которым проводилась операция |
| ACTION\_DATE | DATE | Дата проведения операции |
| ACTION | INT | Идентификатор совершенной операции |
| AMOUNT | FLOAT | Сумма операции |
| ID | INT | Уникальный идентификатор операции |

В данной таблице содержатся данные, необходимые для сохранения истории операций по счетам клиентов.

Таблица 2.7 – Содержание таблицы ACTIONS (Действия доступные для совершения со счетами)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | INT | Уникальный идентификатор операции |
| ACTION\_NAME | NVARCHA2(50) | Название операции |

В данной таблице содержатся данные о доступных для проведения над счетами операциях

Таблица 2.8 – Содержание таблицы DEPOSIT\_CONDITIONS (Информация об условиях доступных депозитов)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | INT | Уникальный идентификатор условий депозита |
| PERCENTAGE\_PER\_YEAR | INT | Процент годовых по условию |
| NAME | NVARCHAR2(50) | Название депозита |
| TERMIN | INT | Срок, на который сумма кладется как депозит |

В данной таблице содержится информация о существующих депозитах и их условиях

Таблица 2.9 – Содержание таблицы DEPOSIT\_STATE (Информация о депозитах, находящихся у банка)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ID | INT | Уникальный идентификатор депозита |
| OWNER | INT | Уникальный идентификатор пользователя, открывшего депозит |
| ACCOUNT\_FOR\_PAY | INT | Счет, на котором хранитя вся сумма депозита |
| DEPOSIT\_TYPE | INT | Уникальный идентификатор условий, по которым заключен депозит |
| CURRENT\_AMOUNT | FLOAT | Сумма на счете в текущий момент |
| START\_DATE | DATE | Дата, когда депозит был создан |

В данной таблице хранится информация необходимая, для отслеживания текущих депозитов, которые хранятся у банка

Основные таблицы, которые обеспечивают минимальную функциональность – это CLIENT\_INFO, CLIENT\_ACCOUNT, ACTIONS, ACCOUNT\_STATS, DEPOSIT\_CONDITIONS, DEPOSIT\_STATE. Данные таблицы соответственно обеспечивают такие функции как: проведение операций по счетам, создание счетов и депозитов, сохранение истории счетов.

2.3 Вывод по разделу

В данном разделе мы начали работу над созданием базы данных банка: создана подключаемая база данных, спроектированы таблицы базы данных, столбцы и типы данных для них, определена модель взаимодействия с системой, а также каким образом будет реализован необходимый функционал при помощи объектов базы данных.

3 Разработка объектов базы данных

**3.1 Табличное пространство**

После создания подключаемой базы данных для нашего проекта и до создания таблиц и заполнения её данными нам необходимо создать табличные пространства – листинг 3.1

create tablespace CLIENT\_DATA

datafile '/opt/oracle/oradata/ORCLCDB1/ORCLPDB1/client\_data.dbf'

size 100m;

create tablespace CLIERK\_DATA

datafile '/opt/oracle/oradata/ORCLCDB1/ORCLPDB1/clerk\_data.dbf'

size 100m;

create temporary tablespace CLIENT\_TMP\_DATA

tempfile '/opt/oracle/oradata/ORCLCDB1/ORCLPDB1/client\_temp\_data.tmp'

size 200m;

create temporary tablespace CLERK\_TMP\_DATA

tempfile '/opt/oracle/oradata/ORCLCDB1/ORCLPDB1/clerk\_temp\_data.tmp'

size 200m;

create tablespace BANK\_DATA

datafile '/opt/oracle/oradata/ORCLCDB1/ORCLPDB1/bank\_data.dbf'

size 200m;

Листинг 3.1 – Создание табличных пространств

После создания пользователей необходимо сделать это табличное пространство для них табличным пространством по умолчанию и выделить квоту.

3.1 Таблицы

3.2.1 Таблицы

Таблица – это структурированное представление данных в базе данных, организованное в виде матрицы из столбцов и строк. Каждый столбец таблицы представляет отдельное поле, имеющее уникальное имя и тип данных, который определяет характер информации, которую оно хранит.

Ключевое поле в таблице играет важную роль, так как оно служит уникальным идентификатором для каждой записи. Это поле обеспечивает однозначное определение каждой строки таблицы и гарантирует уникальность значений в этом поле для каждой записи. Ключевое поле позволяет установить ссылку на конкретные данные и обеспечивает возможность идентификации и доступа к ним.

Из общего можно отметить, что в каждой таблице есть поле, представляющее ID, которое является первичным ключом и заполняется с помощью Identity с шагом 1. Числовые типы данных представлены типами NUMBER, FLOAT, INT. В таблицах CLIENT\_INFO, ACCOUNT\_STATS, DEPOSIT\_STATE присутствует тип DATE для хранения даты. Символьные данные представлены типами nvarchar2 и varchar. На столбцах стоит ограничение целостности «not null».

После создания таблиц они были заполнены небольшим количеством тестовых данных для проверки работоспособности и разработки объектов следующих пунктов.

В данном проекте базы данных содержится девять таблиц, каждая из которых описана в разделе 2. Соответствующий SQL-код для создания этих таблиц приведен в приложении A.

3.3 Процедуры

Хранимая процедура - это именованный фрагмент кода, написанный на языке T-SQL, который может быть создан, изменен и удален с использованием соответствующих операторов (CREATE, ALTER, DROP). Она представляет собой набор инструкций, которые могут быть вызваны и выполнены в базе данных.

Хранимые процедуры могут иметь входные параметры, которые позволяют передавать значения в процедуру при ее вызове, и могут возвращать результаты выполнения. Результаты могут быть в виде целочисленного значения с использованием оператора RETURN, одного или нескольких результирующих наборов с операторами SELECT, или вывода в стандартный поток с помощью оператора PRINT.

Также они могут содержать различные операторы T-SQL, такие как операторы для создания, изменения и удаления данных (DDL, DML и TCL). Они также могут использовать конструкции TRY/CATCH для обработки ошибок, работать с курсорами для обхода результатов запросов, использовать временные таблицы и другие средства, доступные в языке T-SQL.

Вызов хранимой процедуры осуществляется с помощью оператора EXECUTE (или EXEC), за которым следует имя процедуры и необходимые аргументы, если они есть.

Процедуры широко применяются для упрощения и стандартизации выполнения сложных и повторяющихся операций в базе данных, а также для обеспечения безопасности и повышения производительности.

В приложении C представлен листинг SQL-кода для создания процедур, использующихся в курсовом проекте.

3.4 Функции

Функция в PL/SQL - это модуль, который возвращает значение с помощью оператора RETURN. В отличие от процедур, функции могут использовать только входные параметры или не иметь их вовсе, а не аргументы OUT или IN OUT. Вызов функции всегда встроен в исполняемый оператор, может быть частью выражения или использоваться в качестве значения по умолчанию для переменных при их объявлении.

Возвращаемое функцией значение принадлежит определенному типу данных и может быть использовано в выражениях с аналогичным типом данных. Функции способствуют модульному подходу в разработке, позволяя выносить общую логику в отдельные модули. Это упрощает отладку, оптимизацию и обслуживание кода, так как функции могут быть повторно использованы в различных частях программы.

Одним из преимуществ использования функций является возможность разделения кода на более мелкие и самодостаточные модули, что облегчает понимание и поддержку программы. Функции также могут быть оптимизированы независимо от остального кода, что может привести к повышению производительности системы.

В приложении С представлен листинг SQL-кода для создания функций, осуществляющих различные операции.

**3.5 Триггеры базы данных**

Триггер - это особый тип хранимой процедуры, которая не вызывается пользователем напрямую, а автоматически срабатывает при определенных действиях по модификации данных. Он применяется для обеспечения целостности данных и реализации сложной бизнес-логики в базе данных, запускаются сервером базы данных автоматически при попытке изменения данных в таблице, с которой они связаны. Как только происходит действие, вызывающее срабатывание триггера, все модификации данных, выполненные им, рассматриваются как часть текущей транзакции. Это означает, что изменения, произведенные триггером, могут быть подвержены откату, если возникнет ошибка или нарушится целостность данных.

Также они могут выполнять различные операции, такие как вставка, обновление или удаление данных, выполнять сложные проверки, вычисления и взаимодействия с другими таблицами или процедурами в базе данных. Их можно использовать для автоматического обновления связанных данных, регистрации аудита, применения правил бизнес-логики или обеспечения согласованности данных.

Триггеры предоставляют мощный механизм для контроля и автоматизации операций в базе данных, но их использование также требует осторожности и тщательного планирования, чтобы избежать нежелательных побочных эффектов и обеспечить эффективность работы системы.

Для базы данных был создан триггер AFTER\_INSERT\_DEPOSIT\_CONDITION для таблицы DEPOSIT\_CONDITIONS. Триггер выполняется после операцией вставки (AFTER INSERT) для каждой строки (FOR EACH ROW). При помощи данного триггера производится уведомление об успешности вставки. Листинг создания триггера представлен в приложении D.

* 1. Вывод по разделу

В данном разделе было описано и выполнено создание объектов для базы данных веб-сайта, таких как табличное пространство, роли и пользователи, таблицы, триггеры, хранимые процедуры. Также было определено взаимодействие всех этих элементов между собой.

# **4 Описание процедур экспорта и импорта**

4.1 Описание процедуры экспорта

База данных веб-сайта фильмов и сериалов может включать в себя большое количество информации. Здесь и возникает вопрос о том, каким образом можно эффективно экспортировать и импортировать данные. Для хранения данных был выбран JSON-формат. Теперь предстоит разработать соответствующие хранимые процедуры для импорта/экспорта данных. Разберём структуру процедур на примере таблицы CLIENT\_INFO – листинг 4.1.

|  |
| --- |
| create PROCEDURE EXPORT\_CLIENT\_INFO AS  v\_json\_file UTL\_FILE.FILE\_TYPE;  v\_max\_id NUMBER;  p\_file\_path NVARCHAR2(100) := '/opt/oracle/oradata/ORCLCDB1/ORCLPDB1/your\_filename.json';  BEGIN  -- Open the file for writing  v\_json\_file := UTL\_FILE.FOPEN('EXPORT\_DIR', p\_file\_path, 'W');  -- Get the maximum ID  SELECT MAX(id) INTO v\_max\_id FROM CLIENT\_INFO;  -- Write the JSON array start  UTL\_FILE.PUT\_LINE(v\_json\_file, '[');  -- Cursor to fetch data from CLIENT\_INFO  FOR client\_rec IN (SELECT \* FROM CLIENT\_INFO)  LOOP  -- Write JSON object for each row  UTL\_FILE.PUT\_LINE(  v\_json\_file,  '{"name": "' || client\_rec.name || '"' ||  ',"surname": "' || client\_rec.surname || '"' ||  ',"thirdname": "' || client\_rec.thirdname || '"' ||  ',"birth\_date": "' || TO\_CHAR(client\_rec.birth\_date, 'YYYY-MM-DD') || '"' ||  ',"phone\_number": "' || client\_rec.phone\_number || '"}'  );  -- Write comma for all records except the last one  IF client\_rec.id != v\_max\_id THEN  UTL\_FILE.PUT\_LINE(v\_json\_file, ',');  END IF;  END LOOP;  -- Write the JSON array end  UTL\_FILE.PUT\_LINE(v\_json\_file, ']');  -- Close the file  UTL\_FILE.FCLOSE(v\_json\_file);  EXCEPTION  WHEN OTHERS THEN  -- Handle any other unexpected errors  DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);  END EXPORT\_CLIENT\_INFO;  / |

Листинг 4.1 – Процедура для экспорта данных из таблицы VISITS\_TABLE

Предварительно создаём объект Directory, где прописываем путь к папке для хранения данных.

4.2 Описание процедуры импорта

Процедуру импорта рассмотрим также на таблице CLIENT\_INFO. Процедура импорта представлена в листинге 4.2.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE read\_json\_file(P\_FILE\_NAME NVARCHAR2) AS  v\_clob\_data CLOB;  v\_file\_handle UTL\_FILE.FILE\_TYPE;  BEGIN  -- Открытие файла для чтения  v\_file\_handle := UTL\_FILE.FOPEN('JSONDIR', P\_FILE\_NAME, 'r');  BEGIN  LOOP  -- Чтение данных из файла  UTL\_FILE.GET\_LINE(v\_file\_handle, v\_clob\_data);  EXIT WHEN v\_clob\_data IS NULL;  -- Вывод JSON-данных в консоль  DBMS\_OUTPUT.PUT\_LINE(v\_clob\_data);  END LOOP;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  NULL;  END;  -- Закрытие файла  UTL\_FILE.FCLOSE(v\_file\_handle);  -- Вывод завершающего сообщения  DBMS\_OUTPUT.PUT\_LINE('Чтение JSON-данных завершено');  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Ошибка при чтении JSON-данных: ' || SQLERRM);  END read\_json\_file;  / |

Листинг 4.2 – Процедура для импорта данных в таблицу VISITS\_TABLE

В результате импорта мы получим данные, ранее экспортированные из таблицы CLIENT)INFO.

4.3 Вывод по разделу

В этом разделе была рассмотрена проблема заполнения таблиц данными (импорт) и экспорта данных из таблиц, а также предложено её решение с помощью хранимых процедур, реализующих экспорт и импорт данных в/из JSON файл.

# **5 Тестирование производительности**

5.1 Оптимизирование запросов

У нас уже созданы необходимые объекты базы данных, а также таблицы заполнены необходимым количеством данных – рисунок 5.1. Теперь необходимо протестировать, что всё работает, как нужно и как задумано.

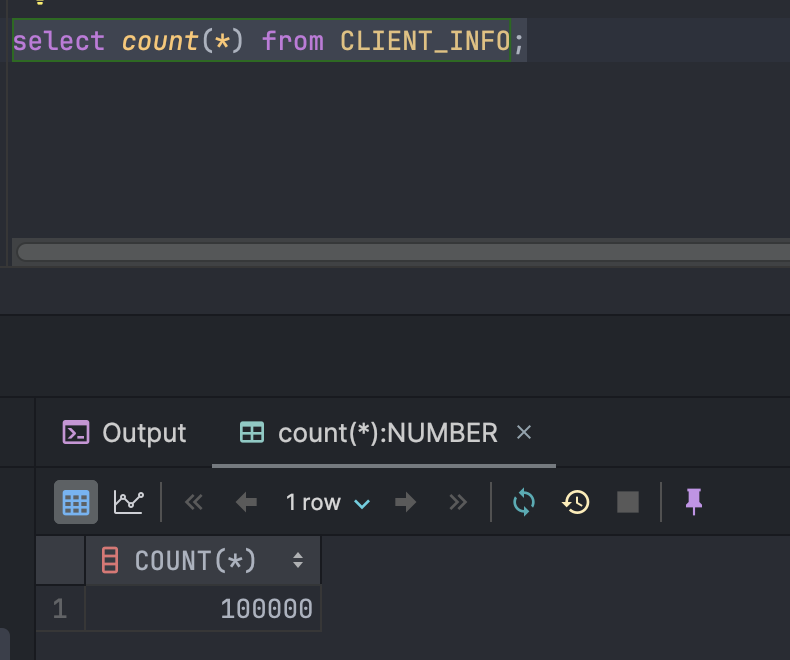


Рисунок 5.1 – Количество строк в таблицe USER\_PROFILE

Для начала проверим, какое количество времени занимает выполнение нашей функции для поиска пользователя по ID– Рисунок 5.2.

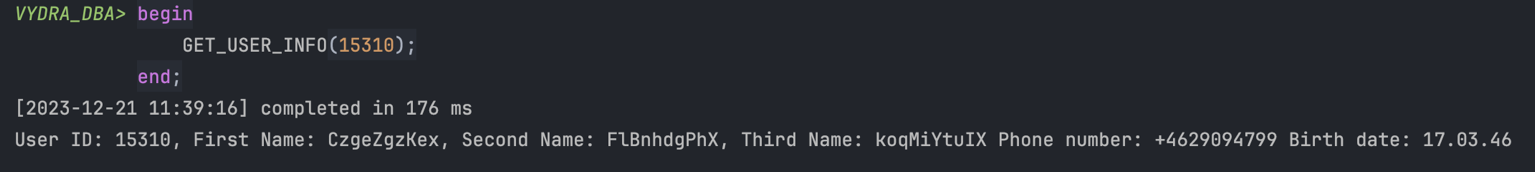


Рисунок 5.2 – Время поиска пользователя

Мы можем заметить, что время ожидания составило всего 0,176 секунды, что является приемлемым результатом. Однако в реальности, в базах данных крупных банков хранятся миллионы записей пользователей, и поиск по ним может занимать значительное время. В данном случае мы просто смоделировали такую ситуацию. Теперь мы выполним простой запрос SELECT для поиска всех пользователей и их счетов, как показано на рисунке 5.3.

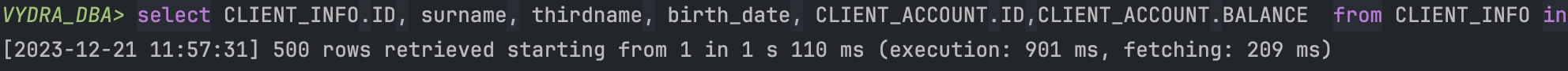


Рисунок 5.3 – Результат SELECT-запроса

Затем мы рассмотрим план выполнения этого запроса, как показано на рисунке 5.4.

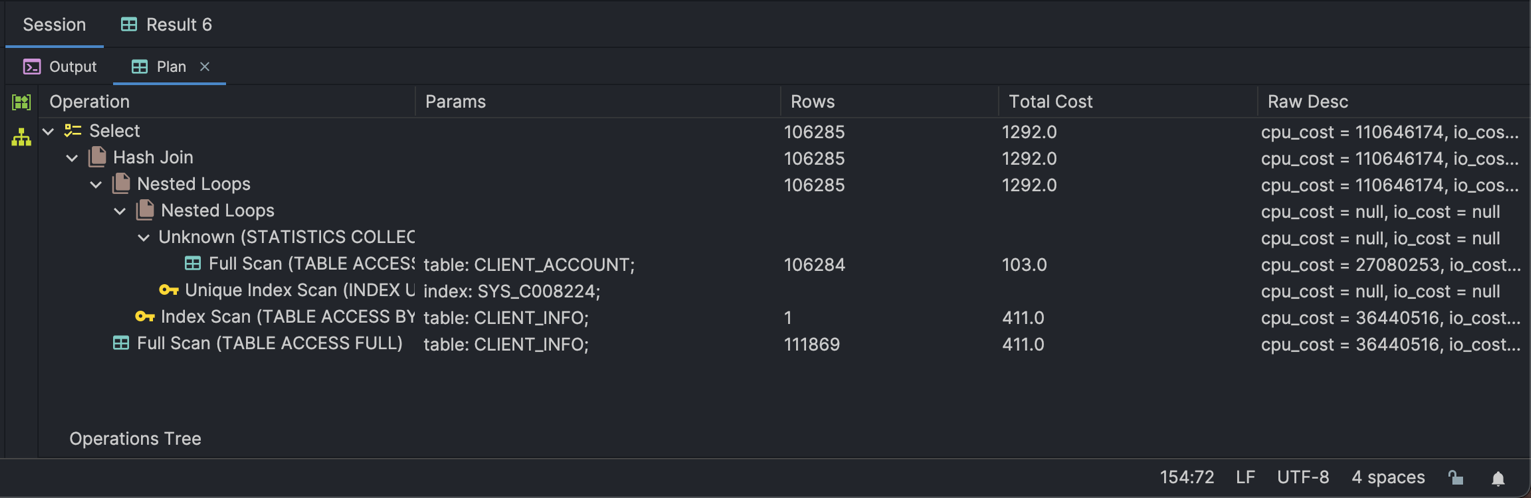


Рисунок 5.4 – План запроса

Теперь попробуем оптимизировать данный запрос с помощью индекса – Листинг 5.1.

CREATE INDEX client\_info\_\_idx

ON client\_info(name, surname, BIRTH\_DATE) ;

Листинг 5.1 – Создание индекса

Создание такого индекса обуславливаем тем, что этот индекс поможет ускорить выполнение запроса, так как он будет использоваться для объединения записей между таблицами USER\_PROFILE и USERS\_REVIEWS\_ON\_MOVIE по столбцу USER\_PROFILE\_ID. Индекс также может улучшить сортировку по столбцу LOGIN.

Теперь проверим результат – рисунок 5.5.

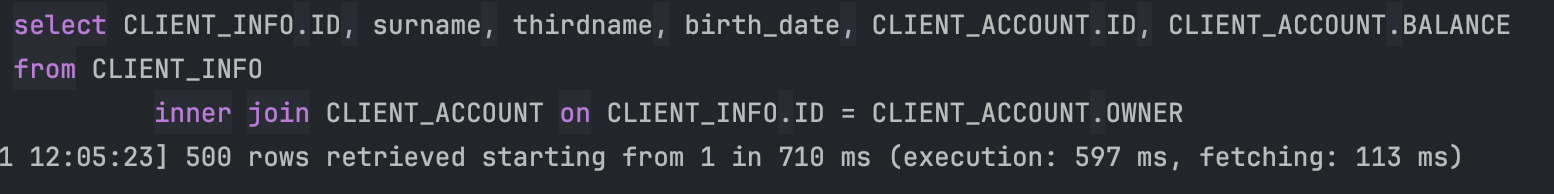


Рисунок 5.5 – Ситуация после создания индекса

После создания индекса время выполнения запроса уменьшилось почти в 6 раз (с 0,18 до 0,011), что свидетельствует о том, что индекс является эффективным.

5.2 Проверка выполнения хранимых процедур

Теперь протестируем основные хранимые процедуры: GET\_USER\_INFO, CLIENT\_CREATION, CREATE\_ACCOUNT\_FOR\_CLIENT, DELETE\_CLIENT. При помощи процедуры CLIENT\_CREATION добавим нового клиента – рисунок 5.6.

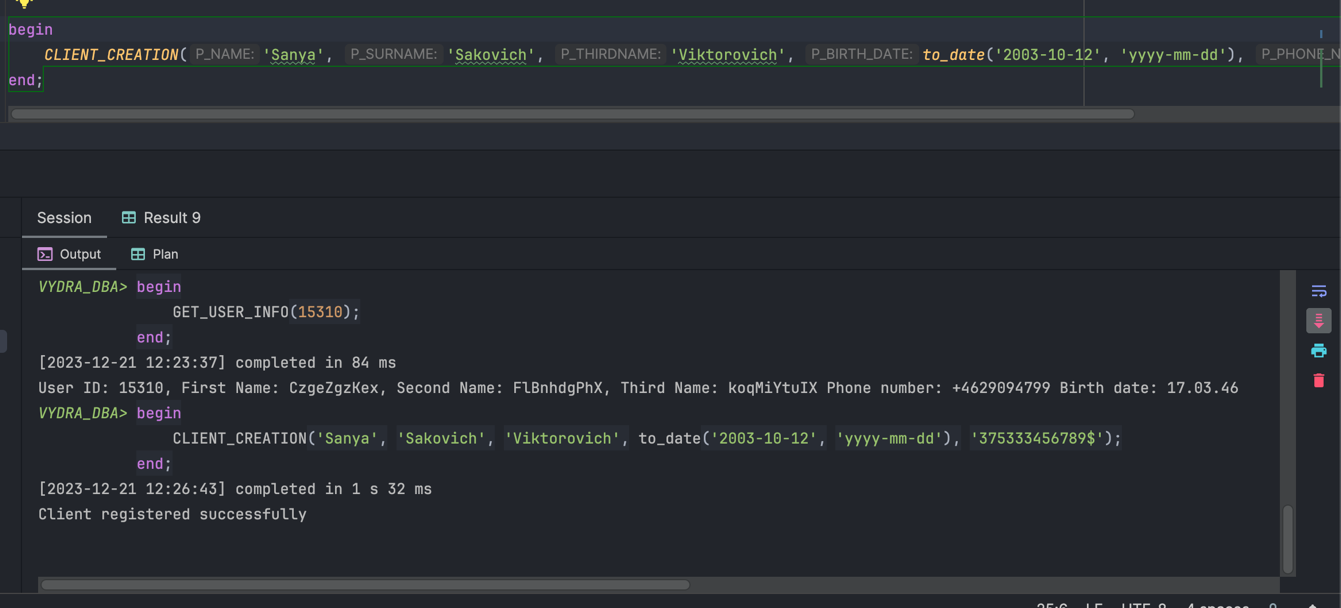


Рисунок 5.6 – Добавление нового пользователя

Далее найдем пользователя по ID. Рисунок 5.7.



Рисунок 5.7 – Поиск пользователя по ID

Создадим счет для клиента. Рисунок 5.8

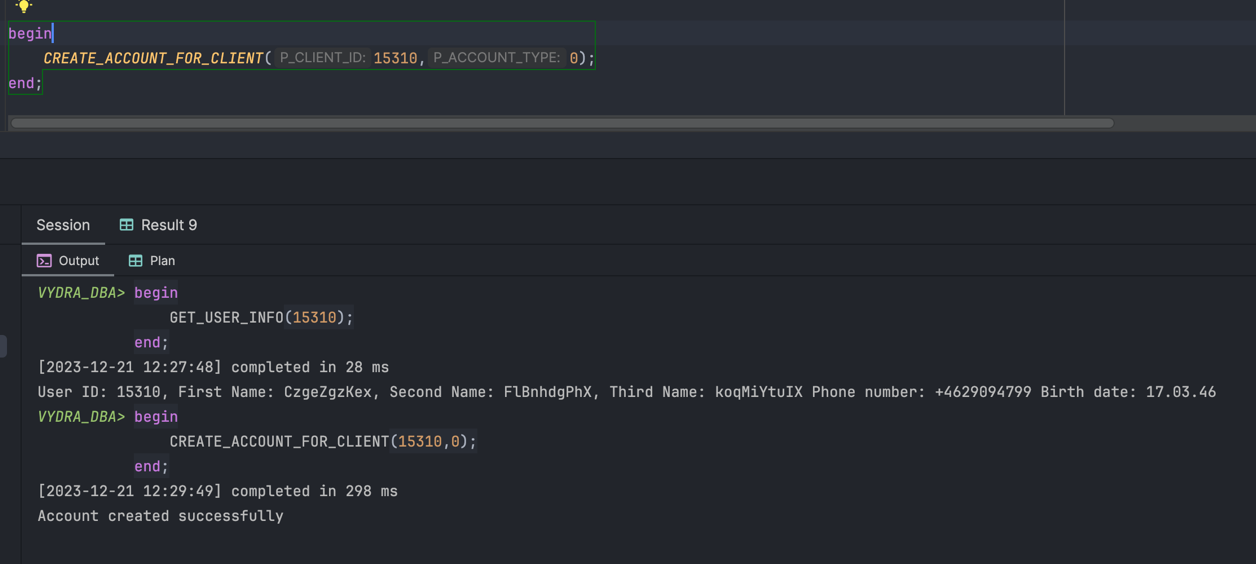


Рисунок 5.8 – Создание счета для клиента

Далее удалим пользователя. Рисунок 5.8.

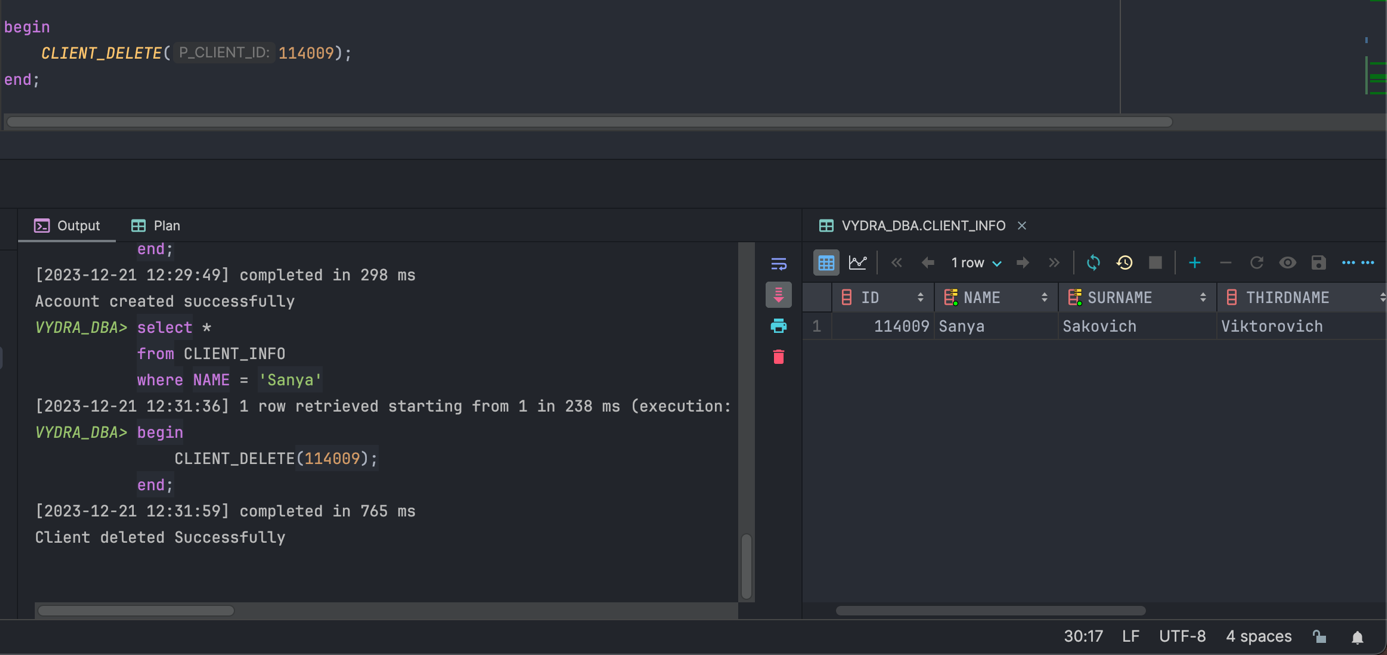


Рисунок 5.8 – Удаление пользователя по ID.

Далее проверим при помощи select запроса удалился ли он. Рисунок 5.9 соответственно.

|  |
| --- |
|  |

Рисунок 5.9 – Проверка удаления пользователя

После select запроса ни одной записи не было возвращено, что свидетельствует о корректном выполнении процедуры CLIENT\_DELETE.

5.2 Вывод по разделу

В данном разделе была протестирована таблица СLIENT\_INFO на вставку и работу с большим количеством данных, создан индекс оптимизации, а также протестированы хранимые процедуры, использующие данную таблицу.

# **6 Описание и применение технологии в БД**

6.1 Описание взаимодействия с технологией

Oracle Enterprise Manager (OEM) представляет собой комплексное программное обеспечение, предназначенное для управления и мониторинга баз данных и приложений, разработанных с использованием продуктов Oracle. Эта технология предоставляет широкий набор инструментов и функций, упрощающих администрирование и оптимизацию ИТ-инфраструктуры на основе технологий Oracle.

Данное приложение имеет широкий спектр применения и может использоваться для анализа различных аспектов работы базы экземпляра базы данных.

После включения технологии enterprise manager к ней можно получить доступ через браузер по адресу loclhost:5500/em. Затем необходимо подключиться к базе данных, чтобы получить доступ к нужным данным. По умолчанию можно подключиться с помощью системных пользователей, например, sys.

Для этого нам необходимо выполнить подключение к своей базе данных через следующее окно – рисунок 6.1.

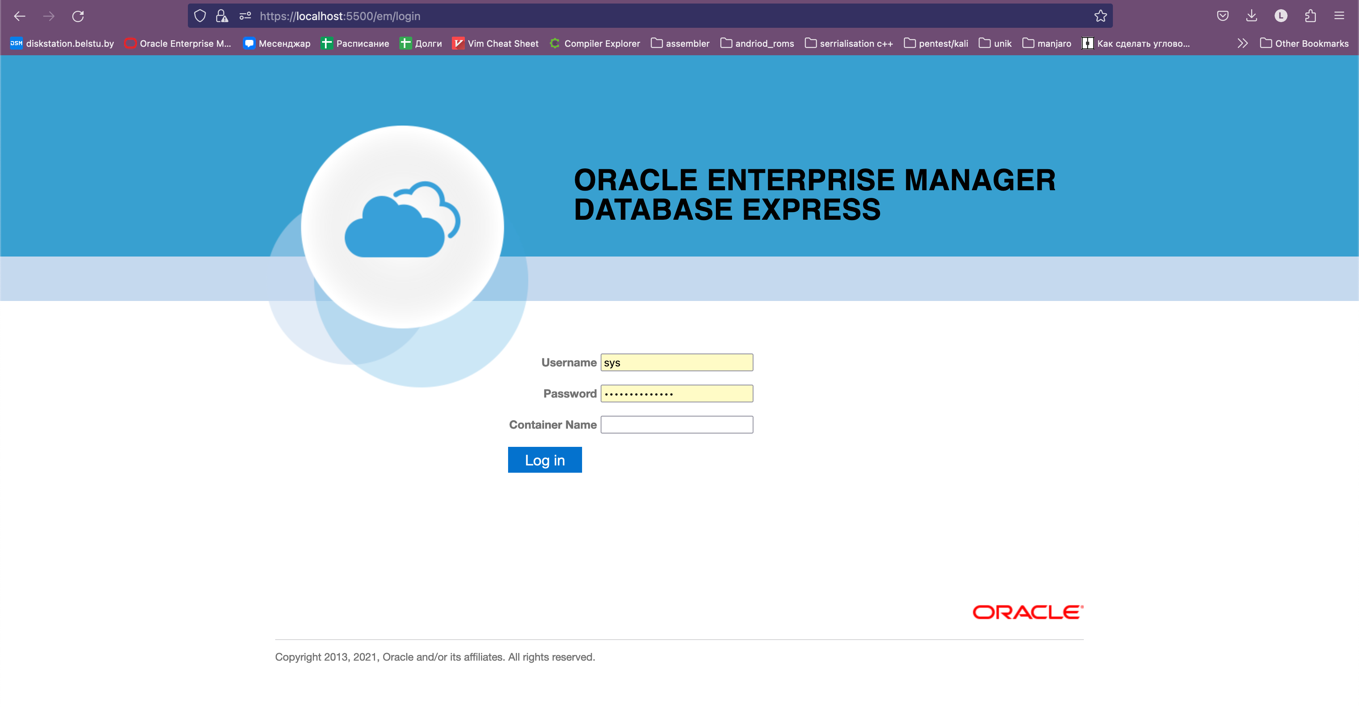


Рисунок 6.1 – Окно подключения

После подключения в основном окне появится статистика по базе данным и последним выполненным процедурам.

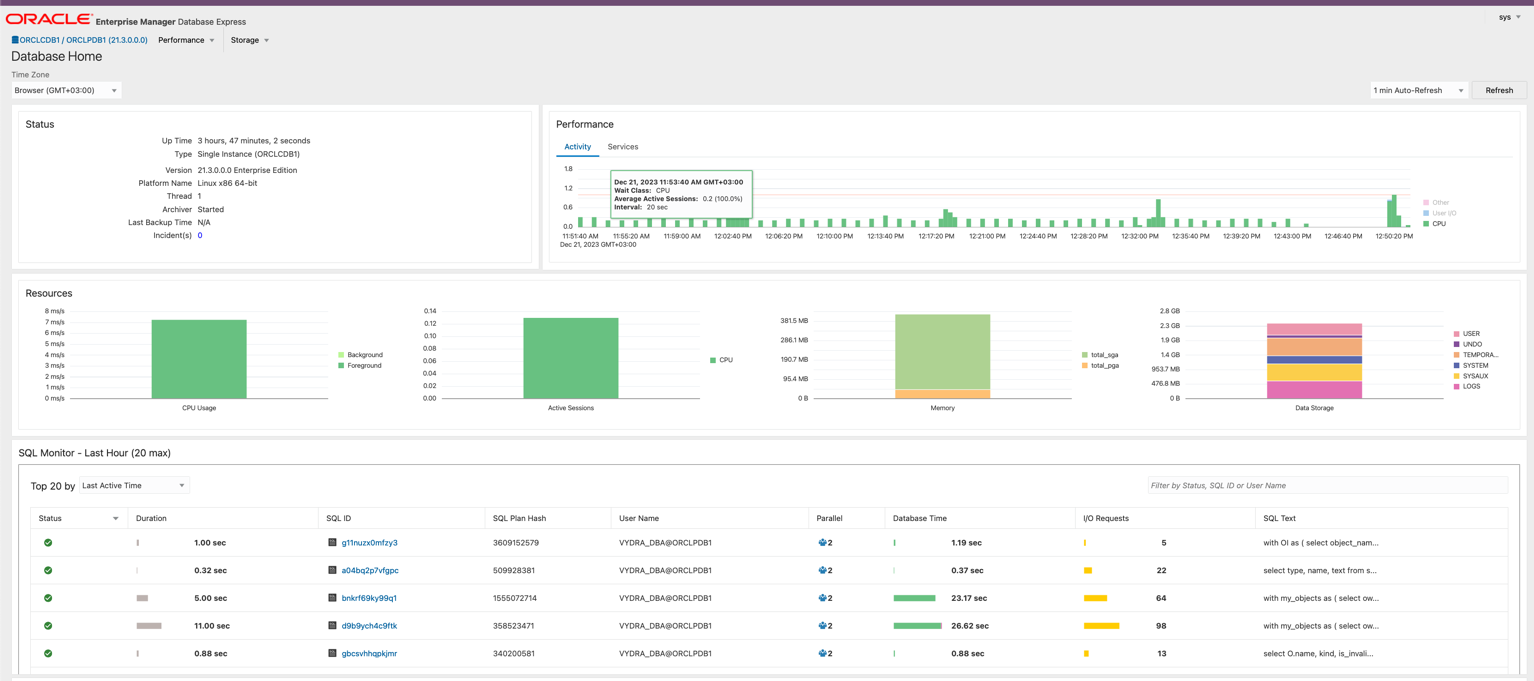


Рисунок 6.2 – Основное окно Oracle enterprise manager

Далее в верхней части после выбора блока Performance можно просмотреть текущее количество ресурсов, которое потребляет база данных

|  |  |
| --- | --- |
|  |  |

Рисунок 6.3 – Окно монитора ресуров

Также можно выбрать блок хранилище и получить статистику по занимаемому объему дискового пространства на каждое табличное пространство

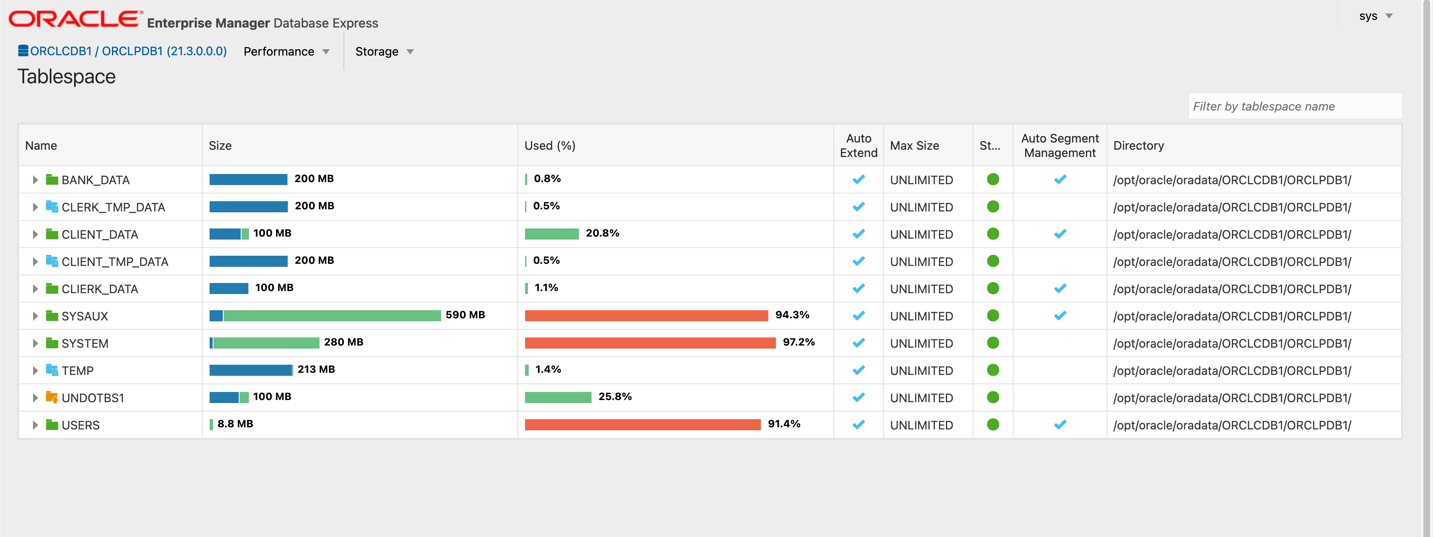


Рисунок 6.4 – Статистика потребления дискового пространства

Также на главной странице Oracle Enterprise manager находится SQL monitor в котором можно посмотреть ресурсы потребляемые процедурами, выполнявшимися в недавнее время. Рисунок 6.5.

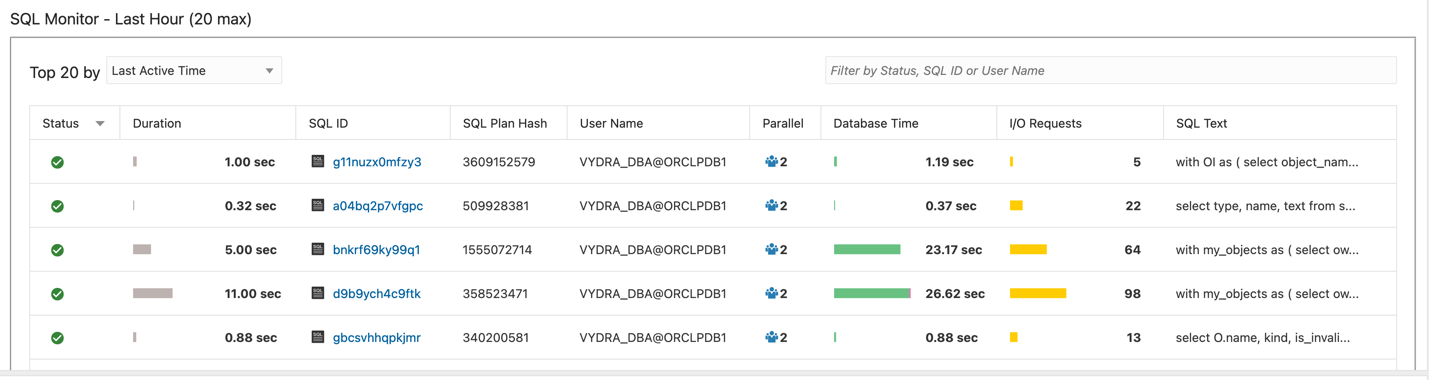


Рисунок 6.5 – SQL monitor

Помимо SQL monitor на главной странице содержится информация об ошибках в базе даных. Это отражено в блоке Incedents (показан на рисунке 6.6).

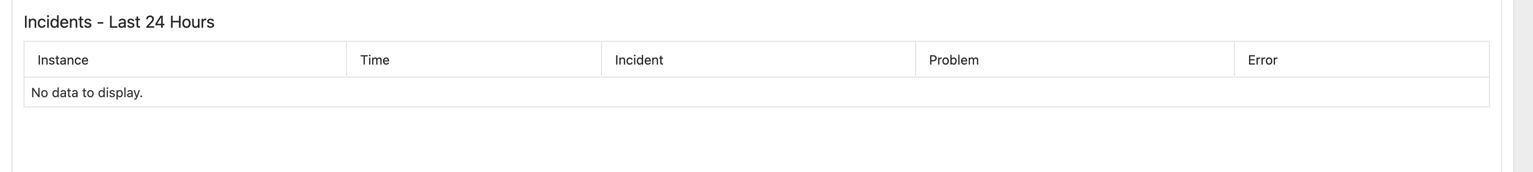


Рисунок 6.6 – Добавление таблицы в набор данных

В результате анализа предоставляемых данных можно отслуживать состояние базы данных и своевременно обновлять инфраструктуру или проводить оптимизацию самой базы данных без прерывания работы.

6.2 Вывод по разделу

Oracle enterprise manager представляет собой мощный инструмент для анализа состояния базы данных. С его помощью можно проводить глубокий и всесторонний анализ работы системы, выявлять неоптимизированные функции и процедуры, что может помочь в принятии стратегических решений.

С помощью интуитивно понятного графического интерфейса приложение упрощает процесс работы с данными, позволяя пользователям без специализированных навыков программирования или аналитики ориентироваться во всех параметрах проихводительности базы данных. Это делает его доступным и полезным инструментом для компаний любого масштаба.

Таким образом, Oracle Enterprise Manager является важным инструментом для технического анализа инфраструктуры, который может помочь компаниям эффективно использовать свои ресурсы и принимать обоснованные стратегические решения.

# **7 Руководство пользователя**

База данных поддерживает 2 вида пользователей: администратор (работник банка) и пользователь (клиент). Каждый из пользователей имеет свое соединение с базой данный. При подключении ему будут доступны процедуры соответствующие его роли. Пользователь может вызвать эти процедуры передав соответствующие параметры в процедуру. Вся информация возвращается в виде консольного вывода DBMS\_OUTPUT. После завершения работы пользователю необходимо завершить сессию и закрыть соединение.

# **Заключение**

В ходе выполнения курсового проекта была спроектирована и реализована инфраструктура базы данных для банка, которая содержит информацию о пользователях, их счетах, операциях по этим счетам с применением СУБД Oracle 21c Enterprise. Были определены и разработаны необходимые объекты базы данных, такие как табличные пространства, таблицы, ограничения целостности, пользователи, триггеры, хранимые процедуры. Также был проведен импорт данных из JSON файлов для заполнения таблиц.

Важным аспектом проекта было применение технологии Oracle Enterprise Manager для анализа данных в базе данных веб-сайта. Были изучены и применены полезные функции, предоставляемые данной технологией, для обеспечения анализа потребляемых ресурсов. Это позволит получить ценные показатели, которые показывают степень оптимизации базы данных и возможную необходимость улучшения инфраструктуры.

Также внимание было уделено производительности базы данных. Проведено тестирование на таблице, содержащей более 100 000 строк, и проведён анализ необходимых изменений в структуре базы данных для обеспечения оптимальной производительности.

В результате успешной реализации курсового проекта была создана база данных для банка с широким функционалом, позволяющим эффективно управлять, пользователями, счетами, депозитами, анализировать активность. Проект демонстрирует преимущества использования СУБД Oracle 19c и технологии Oracle Enterprise Manager для создания мощной и гибкой системы управления базой данных.

# **Список используемых источников**

1. Нистюк О. А. Курс лекций по базам данных, [Электронный ресурс]. – Режим доступа: https://diskstation.belstu.by:5001/. – Дата доступа: 01.09.2023.
2. 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: 01.05.2023.
3. Пользовательское руководство по Oracle Enterprise Manager. Создание подключения, [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/en/middleware/bi/analytics-desktop/bidvd/connect-data-sources.html – Дата доступа: 02.12.2023.
4. [Database Documentation - Oracle Help Center [Электронный ресурс]. – Pежим доступа: https://docs.oracle.com/en/database/ - Дата доступа: 09.12.2023](https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjDm7O5l6CDAxVU_rsIHfTFBBUQFnoECBcQAQ&url=https%3A%2F%2Fdocs.oracle.com%2Fen%2Fdatabase%2F&usg=AOvVaw35aUZFwln6b45zPqbJbquP&opi=89978449)

Приложение A Создание таблиц

|  |
| --- |
| create table CLIENT\_INFO  (  id int generated always as identity primary key,  name nvarchar2(30) not null,  surname nvarchar2(30) not null,  thirdname nvarchar2(30),  birth\_date date not null,  phone\_number nvarchar2(20)  ) tablespace CLIENT\_DATA;  CREATE TABLE CLIENT\_ACCOUNT  (  id int generated always as identity primary key,  owner INT,  is\_locked NUMBER(1) CHECK (is\_locked IN (0, 1)),  account\_type NUMBER(1) CHECK (account\_type IN (0, 1)),  balance NUMBER default 0,  CONSTRAINT fk\_client FOREIGN KEY (owner) REFERENCES CLIENT\_INFO (id)  ) TABLESPACE CLIENT\_DATA;  create table actions  (  id int generated always as identity primary key,  action\_name nvarchar2(50)  ) tablespace BANK\_DATA;  create table ACCOUNT\_STATS  (  id int generated always as identity primary key,  account int,  action\_date date,  action int,  AMOUNT float,  constraint stats\_fk1 foreign key (account) references CLIENT\_ACCOUNT (id),  constraint fk2\_action foreign key (action) references actions (id)  ) tablespace CLIENT\_DATA;  create table deposit\_conditions  (  id int generated always as identity primary key,  procentage\_per\_year float,  name nvarchar2(50),  termin int  ) tablespace BANK\_DATA;  create table deposit\_state  (  id int generated always as identity primary key,  owner int,  current\_amount float,  account\_for\_pay int,  deposit\_type int,  start\_date date,  constraint fk1\_account foreign key (account\_for\_pay) references CLIENT\_ACCOUNT (id),  constraint fk2\_owner foreign key (owner) references CLIENT\_INFO (id),  constraint fk3\_credit\_type foreign key (deposit\_type) references deposit\_conditions (id)  ) tablespace BANK\_DATA;  create table LOGIN\_PASSWORD  (  login varchar(30) primary key,  password varchar(30),  id int,  constraint fk1 foreign key (id) references CLIENT\_INFO (id)  ) tablespace BANK\_DATA;  create table CLERK\_INFO  (  id int generated always as identity primary key,  name nvarchar2(50),  surname nvarchar2(50),  thirdname nvarchar2(50),  birth\_date date  ) tablespace CLIERK\_DATA;  create table CLERK\_LOGIN\_PASSWORD  (  id int,  login nvarchar2(30) primary key,  password nvarchar2(30),  constraint fk1\_cler\_info foreign key (id) references CLERK\_INFO (id)  ) tablespace BANK\_DATA; |

Приложение Б Процедуры и функции

|  |
| --- |
| create PROCEDURE account\_history(  p\_account\_id IN NUMBER,  p\_from\_date IN DATE,  p\_to\_date IN DATE  )  AS  CURSOR c\_account\_history IS  SELECT ACTION\_DATE, ACTION\_NAME, ACCOUNT  FROM ACCOUNT\_STATS  INNER JOIN VYDRA\_DBA.CLIENT\_ACCOUNT CA ON CA.ID = ACCOUNT\_STATS.ACCOUNT  inner join ACTIONS on ACCOUNT\_STATS.ACTION = ACTIONS.ID  WHERE CA.OWNER = p\_account\_id  AND ACTION\_DATE >= p\_from\_date  AND ACTION\_DATE <= p\_to\_date;  v\_operation\_date DATE;  v\_operation NVARCHAR2(50);  v\_account INT;  v\_if\_exists int;  BEGIN  -- Validate input parameters  BEGIN  IF p\_account\_id IS NULL OR p\_from\_date IS NULL OR p\_to\_date IS NULL THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Invalid input parameters. Please provide valid values.');  END IF;  select count(\*) into v\_if\_exists from CLIENT\_ACCOUNT where ID = p\_account\_id;  if v\_if\_exists = 0 then  RAISE\_APPLICATION\_ERROR(-20002, 'Client with given id does not exists');  end if;  -- Check if from\_date is less than or equal to to\_date  IF p\_from\_date > p\_to\_date THEN  RAISE\_APPLICATION\_ERROR(-20002,  'Invalid date range. "From date" should be less than or equal to "To date".');  END IF;  EXCEPTION  WHEN  VALUE\_ERROR THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect type of parameter was passed');  END;  OPEN c\_account\_history;  BEGIN  LOOP  FETCH c\_account\_history INTO v\_operation\_date, v\_operation, v\_account;  EXIT WHEN c\_account\_history%NOTFOUND;  -- Process the selected data as needed  DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || p\_account\_id ||  ', Operation Date: ' || v\_operation\_date ||  ', Operation: ' || v\_operation);  END LOOP;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20003, 'No records found for the specified account and date range.');  END;  CLOSE c\_account\_history;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred: ' || SQLERRM);  END account\_history;  /  create PROCEDURE account\_turnover(p\_user\_id IN INT)  AS  v\_total\_sum NUMBER := 0;  v\_if\_exists number := 0;  CURSOR c\_values\_for\_user IS  SELECT amount  FROM ACCOUNT\_STATS  INNER JOIN CLIENT\_ACCOUNT ON ACCOUNT\_STATS.ACCOUNT = CLIENT\_ACCOUNT.ID  WHERE OWNER = p\_user\_id  AND AMOUNT IS NOT NULL;  BEGIN  select count(\*) into v\_if\_exists from CLIENT\_INFO where ID = p\_user\_id;  if v\_if\_exists = 0 then  RAISE\_APPLICATION\_ERROR(-20001, 'No user found ' || p\_user\_id);  end if;  FOR r\_value IN c\_values\_for\_user  LOOP  v\_total\_sum := v\_total\_sum + r\_value.AMOUNT;  END LOOP;  IF v\_total\_sum IS NULL THEN  RAISE\_APPLICATION\_ERROR(-20001, 'No data found for user ' || p\_user\_id);  ELSE  v\_total\_sum := trunc(v\_total\_sum, 2);  DBMS\_OUTPUT.PUT\_LINE('Total sum for user ' || p\_user\_id || ': ' || v\_total\_sum);  END IF;  EXCEPTION  WHEN  NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20002, 'No data found for user ' || p\_user\_id);  WHEN  OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20003, 'An error occurred: ' || SQLERRM);  END;  /  create PROCEDURE accounts\_amount(p\_client\_id IN INT)  AS  v\_param\_check\_number INT;  v\_amount INT;  BEGIN  -- Проверка на отрицательное значение счета  BEGIN  v\_param\_check\_number := TO\_NUMBER(p\_client\_id);  select count(\*) into v\_param\_check\_number from CLIENT\_INFO where ID = p\_client\_id;  if v\_param\_check\_number = 0 then  RAISE\_APPLICATION\_ERROR(-20002, 'User with given id does not exists');  end if;  -- Проверка на NULL  IF p\_client\_id IS NULL THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect parameters passed');  END IF;  EXCEPTION  WHEN VALUE\_ERROR THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect parameters type');  END;  -- Подсчет количества счетов для текущего пользователя  BEGIN  SELECT COUNT(\*) INTO v\_amount FROM CLIENT\_ACCOUNT WHERE OWNER = p\_client\_id;  IF v\_amount > 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Amount of accounts for current user is: ' || v\_amount);  ELSE  RAISE\_APPLICATION\_ERROR(-20003, 'No records found for the specified account');  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred: ' || SQLERRM);  END;  END accounts\_amount;  /  create PROCEDURE check\_balance(client\_id IN NUMBER)  AS  v\_param\_test NUMBER;  v\_account\_id CLIENT\_ACCOUNT.id%TYPE;  v\_balance CLIENT\_ACCOUNT.balance%TYPE;  CURSOR c\_account\_info IS  SELECT ID, BALANCE  FROM CLIENT\_ACCOUNT  WHERE OWNER = client\_id;  BEGIN  -- Validate that the provided parameter is a positive integer  BEGIN  v\_param\_test := TO\_NUMBER(client\_id);  IF client\_id IS NULL OR client\_id <= 0 OR client\_id != TRUNC(client\_id) THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Invalid client ID provided.');  END IF;  EXCEPTION  WHEN VALUE\_ERROR THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect type of parameter was passed');  END;  -- Select account information  OPEN c\_account\_info;  LOOP  FETCH c\_account\_info INTO v\_account\_id, v\_balance;  v\_balance := trunc(v\_balance, 2);  EXIT WHEN c\_account\_info%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || v\_account\_id || ' Balance: ' || v\_balance);  END LOOP;  CLOSE c\_account\_info;  IF v\_account\_id IS NULL THEN  RAISE\_APPLICATION\_ERROR(-20003, 'No account information found for client ID ' || client\_id);  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred for client ID ' || client\_id || ': ' || SQLERRM);  END check\_balance;  /  create PROCEDURE CLIENT\_INFO\_CHANGING(  p\_client\_id IN INT,  p\_name IN NVARCHAR2,  p\_surname IN NVARCHAR2,  p\_thirdname IN NVARCHAR2,  p\_phone\_number IN NVARCHAR2  ) AS  v\_phone\_check NUMBER;  v\_if\_exists number;  -- Exception for invalid phone number  INVALID\_PHONE EXCEPTION;  PRAGMA EXCEPTION\_INIT (INVALID\_PHONE, -6502);  -- Exception for no rows updated  NO\_ROWS\_UPDATED EXCEPTION;  PRAGMA EXCEPTION\_INIT (NO\_ROWS\_UPDATED, -1422);  BEGIN  select count(\*) into v\_if\_exists from CLIENT\_INFO where ID = p\_client\_id;  if v\_if\_exists != 1 then  RAISE\_APPLICATION\_ERROR(-20001, 'Incorrect client id');  end if;  -- Check if the phone number is a valid number  BEGIN  v\_phone\_check := TO\_NUMBER(p\_phone\_number);  EXCEPTION  WHEN VALUE\_ERROR THEN  RAISE INVALID\_PHONE;  END;  -- Update operations with error handling  BEGIN  IF p\_name IS NOT NULL THEN  UPDATE CLIENT\_INFO SET NAME = p\_name WHERE ID = p\_client\_id;  END IF;  EXCEPTION  WHEN NO\_ROWS\_UPDATED THEN  DBMS\_OUTPUT.PUT\_LINE('No rows updated for NAME');  END;  BEGIN  IF p\_surname IS NOT NULL THEN  UPDATE CLIENT\_INFO SET SURNAME = p\_surname WHERE ID = p\_client\_id;  END IF;  EXCEPTION  WHEN NO\_ROWS\_UPDATED THEN  DBMS\_OUTPUT.PUT\_LINE('No rows updated for SURNAME');  END;  BEGIN  IF p\_thirdname IS NOT NULL THEN  UPDATE CLIENT\_INFO SET THIRDNAME = p\_thirdname WHERE ID = p\_client\_id;  END IF;  EXCEPTION  WHEN NO\_ROWS\_UPDATED THEN  DBMS\_OUTPUT.PUT\_LINE('No rows updated for THIRD NAME');  END;  BEGIN  IF p\_phone\_number IS NOT NULL THEN  UPDATE CLIENT\_INFO SET PHONE\_NUMBER = p\_phone\_number WHERE ID = p\_client\_id;  END IF;  DBMS\_OUTPUT.PUT\_LINE('Client edited successfully');  EXCEPTION  WHEN NO\_ROWS\_UPDATED THEN  DBMS\_OUTPUT.PUT\_LINE('No rows updated for PHONE NUMBER');  WHEN INVALID\_PHONE THEN  DBMS\_OUTPUT.PUT\_LINE('Invalid phone number');  END;  END CLIENT\_INFO\_CHANGING;  /  create PROCEDURE COUNT\_DEPOSIT\_PROFIT(p\_account\_id IN INT, p\_owner IN INT) AS  v\_deposit\_type INT;  v\_basic\_amount FLOAT;  v\_recounts INT;  v\_percentage FLOAT;  v\_counter INT := 1;  BEGIN  -- Обработка NO\_DATA\_FOUND, если запись в DEPOSIT\_STATE не найдена  BEGIN  SELECT DEPOSIT\_TYPE, CURRENT\_AMOUNT  INTO v\_deposit\_type, v\_basic\_amount  FROM DEPOSIT\_STATE  WHERE OWNER = p\_owner  AND ACCOUNT\_FOR\_PAY = p\_account\_id;  -- Обработка NO\_DATA\_FOUND, если запись в DEPOSIT\_CONDITIONS не найдена  BEGIN  SELECT TERMIN, PROCENTAGE\_PER\_YEAR  INTO v\_recounts, v\_percentage  FROM DEPOSIT\_CONDITIONS  WHERE ID = v\_deposit\_type;  -- Проверка, что процент неотрицателен  IF v\_percentage < 0 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Invalid percentage value');  END IF;  -- Обработка ZERO\_DIVIDE, чтобы избежать деления на 0  IF v\_percentage = 0 THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Percentage cannot be zero');  END IF;  LOOP  v\_basic\_amount := v\_basic\_amount + (v\_basic\_amount \* (v\_percentage / 100));  v\_counter := v\_counter + 1;  -- Выход из цикла при достижении v\_recounts  EXIT WHEN v\_counter > v\_recounts;  END LOOP;  DBMS\_OUTPUT.PUT\_LINE('Total amount at the end of the deposit term will be: ' || v\_basic\_amount);  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('No data found in DEPOSIT\_CONDITIONS for the specified deposit type');  WHEN ZERO\_DIVIDE THEN  DBMS\_OUTPUT.PUT\_LINE('Percentage value is zero, causing division by zero');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);  END;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('No data found in DEPOSIT\_STATE for the specified account and owner');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);  END;  END COUNT\_DEPOSIT\_PROFIT;  /  create PROCEDURE get\_user\_info(p\_user\_id IN NUMBER)  AS  v\_first\_name nVARCHAR2(50);  v\_second\_name nVARCHAR2(50);  v\_third\_name nVARCHAR2(50);  v\_phone\_number nvarchar2(50);  v\_birth\_date date;  v\_param\_test NUMBER;  BEGIN  -- Validate that the provided parameter is a positive integer  BEGIN  v\_param\_test := TO\_NUMBER(p\_user\_id);  IF p\_user\_id IS NULL OR p\_user\_id <= 0 OR p\_user\_id != TRUNC(p\_user\_id) THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Invalid user ID provided.');  END IF;  EXCEPTION  WHEN VALUE\_ERROR THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect type of parameter was passed');  END;  -- Select user information  BEGIN  SELECT CLIENT\_INFO.NAME,  CLIENT\_INFO.SURNAME,  CLIENT\_INFO.THIRDNAME,  CLIENT\_INFO.PHONE\_NUMBER,  CLIENT\_INFO.BIRTH\_DATE  INTO v\_first\_name, v\_second\_name, v\_third\_name, v\_phone\_number, v\_birth\_date  FROM CLIENT\_INFO  WHERE ID = p\_user\_id;  -- Check if the user has a third name  IF v\_third\_name IS NOT NULL THEN  DBMS\_OUTPUT.PUT\_LINE('User ID: ' || p\_user\_id || ', First Name: ' || v\_first\_name ||  ', Second Name: ' || v\_second\_name || ', Third Name: ' || v\_third\_name ||  ' Phone number: +' || v\_phone\_number || ' Birth date: ' || to\_char(v\_birth\_date));  ELSE  DBMS\_OUTPUT.PUT\_LINE('User ID: ' || p\_user\_id || ', First Name: ' || v\_first\_name ||  ', Second Name: ' || v\_second\_name || ', No Third Name' || ' Phone number: +' ||  v\_phone\_number || ' Birth date: ' || to\_char(v\_birth\_date));  END IF;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20003, 'User with ID ' || p\_user\_id || ' not found.');  END;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred for User ID ' || p\_user\_id || ': ' || SQLERRM);  END get\_user\_info;  /  create PROCEDURE LOGIN\_CLIENT(  p\_login IN NVARCHAR2,  p\_password IN NVARCHAR2  ) AS  v\_is\_exists NUMBER;  BEGIN  -- Check if the login and password exist in LOGIN\_PASSWORD table  SELECT COUNT(\*)  INTO v\_is\_exists  FROM LOGIN\_PASSWORD  WHERE LOGIN = p\_login  AND PASSWORD = p\_password;  -- Handle the case when login or password is incorrect  IF v\_is\_exists = 0 THEN  -- Raise an exception for incorrect login or password  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect login or password');  ELSE  DBMS\_OUTPUT.PUT\_LINE('Client logged in successfully');  END IF;  EXCEPTION  WHEN OTHERS THEN  -- Handle any other unexpected errors  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred: ' || SQLERRM);  END LOGIN\_CLIENT;  /  create PROCEDURE operations\_amount(  p\_from\_date IN DATE,  p\_to\_date IN DATE,  p\_user\_id IN INT,  p\_account\_id IN INT  )  AS  v\_operations\_count INT;  BEGIN  -- Проверка на отрицательное значение счета  IF p\_from\_date IS NULL OR p\_to\_date IS NULL OR p\_user\_id IS NULL OR p\_account\_id IS NULL THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect parameters passed');  END IF;  SELECT count(\*)  into v\_operations\_count  FROM ACCOUNT\_STATS  INNER JOIN VYDRA\_DBA.CLIENT\_ACCOUNT CA ON CA.ID = ACCOUNT\_STATS.ACCOUNT  WHERE CA.OWNER = p\_user\_id  and ACCOUNT\_STATS.ACCOUNT = p\_account\_id  AND ACTION\_DATE >= p\_from\_date  AND ACTION\_DATE <= p\_to\_date;  IF v\_operations\_count > 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Number of operations for user ' || p\_user\_id || ' and account ' || p\_account\_id || ': ' ||  v\_operations\_count);  ELSE  RAISE\_APPLICATION\_ERROR(-20003, 'No data found for the specified criteria.');  END IF;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20003, 'No data found for the specified criteria.');  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred: ' || SQLERRM);  END operations\_amount;  /  create PROCEDURE refill\_account(p\_account\_id IN NUMBER, p\_amount IN NUMBER)  AS  v\_param\_check NUMBER;  BEGIN  -- Проверка на NULL  BEGIN  IF p\_account\_id IS NULL OR p\_amount IS NULL THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect parameters passed');  END IF;  EXCEPTION  WHEN VALUE\_ERROR THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect parameters type');  END;  -- Попытка преобразования параметров к числовому типу  BEGIN  -- Проверка на отрицательное значение счета  IF p\_amount <= 0 THEN  RAISE\_APPLICATION\_ERROR(-20003, 'Amount must be greater than 0');  END IF;  -- Проверка на существование счета  BEGIN  SELECT COUNT(\*) into v\_param\_check FROM CLIENT\_ACCOUNT WHERE ID = p\_account\_id and IS\_LOCKED = 0;  IF v\_param\_check = 1 THEN  UPDATE CLIENT\_ACCOUNT SET BALANCE = BALANCE + p\_amount WHERE ID = p\_account\_id;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Account refilled successfully');  ELSE  RAISE\_APPLICATION\_ERROR(-20004, 'Account does not exist or it is locked');  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while checking account existence: ' || SQLERRM);  END;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred: ' || SQLERRM);  END;  END refill\_account;  /  create PROCEDURE REGISTER\_CLIENT(  p\_login IN nvarchar2,  p\_password IN NVARCHAR2,  p\_user\_id IN INT  ) AS  v\_client\_count INT;  v\_login\_taken INT;  BEGIN  -- Check if the client with the given ID exists  SELECT COUNT(\*)  INTO v\_client\_count  FROM CLIENT\_INFO  WHERE CLIENT\_INFO.ID = p\_user\_id;  -- Check if the login is already taken  SELECT COUNT(\*)  INTO v\_login\_taken  FROM LOGIN\_PASSWORD  WHERE p\_login = LOGIN;  -- Handle the case when the client with the given ID does not exist  IF v\_client\_count != 0 AND v\_login\_taken = 0 THEN  -- Insert login and password for the client  INSERT INTO LOGIN\_PASSWORD (LOGIN, PASSWORD, ID)  VALUES (p\_login, p\_password, p\_user\_id);  DBMS\_OUTPUT.PUT\_LINE('Client registered successfully');  ELSIF v\_login\_taken > 0 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'This login is already taken');  ELSE  -- Raise an exception when the client with the given ID does not exist  RAISE\_APPLICATION\_ERROR(-20001, 'Incorrect ID: No such client');  RETURN;  END IF;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  -- Handle the case when there is no data found during the SELECT statement  RAISE\_APPLICATION\_ERROR(-20002, 'No such client');  WHEN VALUE\_ERROR THEN  -- Handle the case when a value error occurs (e.g., incorrect parameter type)  RAISE\_APPLICATION\_ERROR(-20003, 'Incorrect parameter type');  WHEN OTHERS THEN  -- Handle any other unexpected errors  RAISE\_APPLICATION\_ERROR(-20004, 'An error occurred: ' || SQLERRM);  END REGISTER\_CLIENT;  /  create PROCEDURE transfer(  p\_sender\_account IN NUMBER,  p\_receiver\_account IN NUMBER,  p\_sender\_id IN NUMBER,  p\_amount IN NUMBER  )  AS  v\_check\_sender NUMBER;  v\_check\_receiver NUMBER;  v\_check\_parameter NUMBER;  BEGIN  -- Validate that the parameters are valid numbers  BEGIN  v\_check\_parameter := TO\_NUMBER(p\_sender\_account);  v\_check\_parameter := TO\_NUMBER(p\_receiver\_account);  v\_check\_parameter := TO\_NUMBER(p\_sender\_id);  v\_check\_parameter := TO\_NUMBER(p\_amount);  EXCEPTION  WHEN VALUE\_ERROR THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect parameter type');  END;  -- Check if any parameter is NULL  IF p\_sender\_account IS NULL OR p\_receiver\_account IS NULL OR p\_sender\_id IS NULL OR p\_amount IS NULL THEN  RAISE\_APPLICATION\_ERROR(-20002, 'One or more parameters were not correctly passed.');  END IF;  -- Check sender account  SELECT COUNT(\*)  INTO v\_check\_sender  FROM CLIENT\_ACCOUNT  WHERE ID = p\_sender\_account  AND OWNER = p\_sender\_id  AND BALANCE >= p\_amount  and IS\_LOCKED = 0;  -- Check receiver account  SELECT COUNT(\*)  INTO v\_check\_receiver  FROM CLIENT\_ACCOUNT  WHERE ID = p\_receiver\_account  and IS\_LOCKED = 0;  -- Validate sender and receiver  IF v\_check\_sender = 0 THEN  RAISE\_APPLICATION\_ERROR(-20003, 'Invalid sender data or locked account');  END IF;  IF v\_check\_receiver = 0 THEN  RAISE\_APPLICATION\_ERROR(-20004, 'Invalid receiver data or locked account');  END IF;  -- Update balances  BEGIN  UPDATE CLIENT\_ACCOUNT  SET BALANCE = BALANCE + p\_amount  WHERE ID = p\_receiver\_account;  UPDATE CLIENT\_ACCOUNT  SET BALANCE = BALANCE - p\_amount  WHERE ID = p\_sender\_account;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Error updating account balances: ' || SQLERRM);  END;  -- Insert into ACCOUNT\_STATS  BEGIN  INSERT INTO ACCOUNT\_STATS (account, action\_date, action, AMOUNT)  VALUES (p\_sender\_account, SYSDATE, 3, p\_amount);  INSERT INTO ACCOUNT\_STATS (account, action\_date, action, AMOUNT)  VALUES (p\_receiver\_account, SYSDATE, 1, p\_amount);  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Error inserting into ACCOUNT\_STATS: ' || SQLERRM);  END;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Successfully transferred ' || p\_amount ||  ' from account ' || p\_sender\_account ||  ' to ' || p\_receiver\_account);  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred: ' || SQLERRM);  END transfer;  /  create PROCEDURE withdrawal(p\_account\_id IN NUMBER, p\_amount IN NUMBER, p\_client\_id in int)  AS  BEGIN  -- Проверка на NULL  BEGIN  IF p\_account\_id IS NULL OR p\_amount IS NULL THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect parameters passed');  END IF;  EXCEPTION  WHEN VALUE\_ERROR THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect parameters type');  END;  -- Проверка на отрицательное значение счета  IF p\_amount <= 0 THEN  RAISE\_APPLICATION\_ERROR(-20003, 'Amount must be greater than 0');  END IF;  -- Проверка на существование счета  DECLARE  v\_account\_count NUMBER;  BEGIN  SELECT COUNT(\*)  INTO v\_account\_count  FROM CLIENT\_ACCOUNT  WHERE ID = p\_account\_id and IS\_LOCKED = 0 and OWNER = p\_client\_id;  IF v\_account\_count = 1 THEN  UPDATE CLIENT\_ACCOUNT SET BALANCE = BALANCE - p\_amount WHERE ID = p\_account\_id;  insert into ACCOUNT\_STATS(account, action\_date, action, amount) values (p\_account\_id, sysdate, 2, p\_amount);  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Money was withdrawn successfully');  ELSE  RAISE\_APPLICATION\_ERROR(-20004, 'Account does not exist or it is locked');  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while checking account existence: ' || SQLERRM);  END;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred: ' || SQLERRM);  END withdrawal;  /  create PROCEDURE CHANGE\_CLIENT\_ACCOUNT\_TYPE(p\_user\_id IN INT, p\_client\_account IN INT) AS  v\_if\_exists INT;  v\_account\_type INT;  BEGIN  -- Check if the account with the given ID and client ID exists  SELECT COUNT(\*)  INTO v\_if\_exists  FROM CLIENT\_ACCOUNT  WHERE ID = p\_client\_account  AND OWNER = p\_user\_id;  -- Handle the case when the account does not exist  IF v\_if\_exists = 0 THEN  -- Raise an exception for non-existent client or account  RAISE\_APPLICATION\_ERROR(-20001, 'Client with ID ' || p\_user\_id || ' or account with ID ' || p\_client\_account ||  ' does not exist');  RETURN;  ELSE  -- Get the current account type  SELECT ACCOUNT\_TYPE  INTO v\_account\_type  FROM CLIENT\_ACCOUNT  WHERE ID = p\_client\_account  AND OWNER = p\_user\_id;  -- Toggle the account type  IF v\_account\_type = 0 THEN  UPDATE CLIENT\_ACCOUNT  SET ACCOUNT\_TYPE = 1  WHERE ID = p\_client\_account  AND OWNER = p\_user\_id;  ELSE  UPDATE CLIENT\_ACCOUNT  SET ACCOUNT\_TYPE = 0  WHERE ID = p\_client\_account  AND OWNER = p\_user\_id;  END IF;  DBMS\_OUTPUT.PUT\_LINE('Account type changed successfully');  END IF;  EXCEPTION  WHEN OTHERS THEN  -- Handle any other unexpected errors  RAISE\_APPLICATION\_ERROR(-20002, 'An error occurred: ' || SQLERRM);  END CHANGE\_CLIENT\_ACCOUNT\_TYPE;  /  create PROCEDURE CLERK\_LOGIN(  p\_login IN nvarchar2,  p\_password IN NVARCHAR2  ) AS  v\_is\_exists NUMBER;  BEGIN  -- Check for null parameters  IF p\_login IS NULL OR p\_password IS NULL THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect parameters passed');  END IF;  -- Check login and password in CLERK\_LOGIN\_PASSWORD table  SELECT COUNT(\*)  INTO v\_is\_exists  FROM CLERK\_LOGIN\_PASSWORD  WHERE LOGIN = p\_login  AND PASSWORD = p\_password;  -- Handle incorrect login or password  IF v\_is\_exists != 1 THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect login or password');  ELSE  DBMS\_OUTPUT.PUT\_LINE('Worker logged in successfully');  END IF;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect login or password');  WHEN TOO\_MANY\_ROWS THEN  RAISE\_APPLICATION\_ERROR(-20003,  'Multiple records found for the same login and password. Data integrity issue.');  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred: ' || SQLERRM);  END CLERK\_LOGIN;  /  create PROCEDURE CLIENT\_CREATION(  p\_name IN NVARCHAR2,  p\_surname IN NVARCHAR2,  p\_thirdname IN NVARCHAR2 DEFAULT NULL,  p\_birth\_date IN DATE,  p\_phone\_number NVARCHAR2  )  AS  v\_if\_exists int;  BEGIN  -- Check if the phone number is a valid number  -- Insert into CLIENT\_INFO  IF p\_name IS NOT NULL AND p\_surname IS NOT NULL AND p\_birth\_date IS NOT NULL AND p\_phone\_number IS NOT NULL and  (extract(year from p\_birth\_date) <= (extract(year from sysdate) - 18)) and length(p\_phone\_number) = 13 and  REGEXP\_LIKE(p\_phone\_number, '^[0-9]') THEN  select count(\*)  into v\_if\_exists  from CLIENT\_INFO  where NAME = p\_name  and SURNAME = p\_surname  and BIRTH\_DATE = p\_birth\_date  and PHONE\_NUMBER = p\_phone\_number;  --INSERT INTO CLIENT\_INFO(name, surname, thirdname, birth\_date, phone\_number)  --VALUES (p\_name, p\_surname, NVL(p\_thirdname, ''), p\_birth\_date, p\_phone\_number);  --DBMS\_OUTPUT.PUT\_LINE('Client registered successfully');  ELSE  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect client data');  END IF;  if v\_if\_exists = 0 then  INSERT INTO CLIENT\_INFO(name, surname, thirdname, birth\_date, phone\_number)  VALUES (p\_name, p\_surname, NVL(p\_thirdname, ''), p\_birth\_date, p\_phone\_number);  DBMS\_OUTPUT.PUT\_LINE('Client registered successfully');  else  RAISE\_APPLICATION\_ERROR(-20001, 'Client with given parametres already exists');  end if;  EXCEPTION  WHEN VALUE\_ERROR THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Invalid value for phone number');  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred during client creation: ' || SQLERRM);  END client\_creation;  /  create PROCEDURE CLIENT\_DELETE(p\_client\_id IN INT) AS  v\_is\_client\_exists INT;  v\_accounts\_avalible\_for\_delete int;  v\_total\_avccounts int;  -- Exception for no rows found  NO\_ROWS\_FOUND EXCEPTION;  PRAGMA EXCEPTION\_INIT (NO\_ROWS\_FOUND, -20001);  -- Exception for too many rows found  TOO\_MANY\_ROWS EXCEPTION;  PRAGMA EXCEPTION\_INIT (TOO\_MANY\_ROWS, -20002);  -- Exception for general error during deletion  DELETE\_ERROR EXCEPTION;  PRAGMA EXCEPTION\_INIT (DELETE\_ERROR, -20003);  BEGIN  -- Check if the client exists  BEGIN  SELECT COUNT(\*)  INTO v\_is\_client\_exists  FROM CLIENT\_INFO  WHERE ID = p\_client\_id;  IF v\_is\_client\_exists != 1 THEN  RAISE\_APPLICATION\_ERROR(-20003, 'Incorrect client id ');  END IF;  EXCEPTION  WHEN NO\_ROWS\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20002, 'No client found with the specified ID.');  WHEN TOO\_MANY\_ROWS THEN  RAISE\_APPLICATION\_ERROR(-20003, 'Multiple clients found with the same ID. Data integrity issue.');  END;  -- Delete the client  BEGIN  select count(\*)  into v\_accounts\_avalible\_for\_delete  from CLIENT\_ACCOUNT  inner join DEPOSIT\_STATE on CLIENT\_ACCOUNT.ID = DEPOSIT\_STATE.ACCOUNT\_FOR\_PAY  where CLIENT\_ACCOUNT.BALANCE = 0  and DEPOSIT\_STATE.CURRENT\_AMOUNT = 0  and CLIENT\_ACCOUNT.OWNER = p\_client\_id;  select count(\*) into v\_total\_avccounts from CLIENT\_ACCOUNT where OWNER = p\_client\_id;  if v\_accounts\_avalible\_for\_delete = v\_total\_avccounts then  Delete from CLIENT\_ACCOUNT where OWNER = p\_client\_id;  DELETE  FROM CLIENT\_INFO  WHERE ID = p\_client\_id;  DBMS\_OUTPUT.PUT\_LINE('Client deleted Successfully');  else  RAISE\_APPLICATION\_ERROR(-20001, 'Client have money on theirs account, delete operation unavailable ');  end if;  EXCEPTION  WHEN  OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred during client deletion: ' || SQLERRM);  -- Optionally, you can log or handle the error as needed.  END;  END CLIENT\_DELETE;  /  create PROCEDURE CREATE\_ACCOUNT\_FOR\_CLIENT(p\_client\_id IN INT, p\_account\_type IN INT) AS  v\_is\_client\_exists INT;  BEGIN  -- Check if the client with the given ID exists  SELECT COUNT(\*)  INTO v\_is\_client\_exists  FROM CLIENT\_INFO  WHERE ID = p\_client\_id;  -- Handle the case when the client does not exist  IF v\_is\_client\_exists = 0 THEN  -- Raise an exception for non-existent client  RAISE\_APPLICATION\_ERROR(-20001, 'Client with ID ' || p\_client\_id || ' does not exist');  RETURN;  END IF;  -- Insert a new account for the client  INSERT INTO CLIENT\_ACCOUNT (OWNER, IS\_LOCKED, ACCOUNT\_TYPE, BALANCE)  VALUES (p\_client\_id, 0, p\_account\_type, 0);  DBMS\_OUTPUT.PUT\_LINE('Account created successfully');  EXCEPTION  WHEN VALUE\_ERROR THEN  -- Handle the case when a value error occurs (e.g., incorrect parameter type)  RAISE\_APPLICATION\_ERROR(-20002, 'Incorrect parameter type');  WHEN OTHERS THEN  -- Handle any other unexpected errors  RAISE\_APPLICATION\_ERROR(-20003, 'An error occurred: ' || SQLERRM);  END CREATE\_ACCOUNT\_FOR\_CLIENT;  /  create PROCEDURE CREATE\_DEPOSIT(  p\_user\_id IN INT,  p\_deposit\_condition IN INT,  p\_amount IN NUMBER  ) AS  v\_account\_id INT;  v\_if\_exists INT;  BEGIN  -- Проверка на существование пользователя  SELECT COUNT(\*) INTO v\_if\_exists FROM CLIENT\_INFO WHERE ID = p\_user\_id;  IF v\_if\_exists != 1 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Incorrect user id');  END IF;  IF p\_deposit\_condition IS NOT NULL AND p\_amount IS NOT NULL THEN  -- Вставка записи в таблицу CLIENT\_ACCOUNT  INSERT INTO CLIENT\_ACCOUNT(owner, is\_locked, account\_type, balance)  VALUES (p\_user\_id, 0, 1, p\_amount);  END IF;  -- Получение ID созданного счета  SELECT ID  INTO v\_account\_id  FROM CLIENT\_ACCOUNT  WHERE ROWNUM = 1  AND ACCOUNT\_TYPE = 1  ORDER BY ID DESC;  -- Вставка записи в таблицу DEPOSIT\_STATE  INSERT INTO DEPOSIT\_STATE(owner, current\_amount, account\_for\_pay, deposit\_type, start\_date)  VALUES (p\_user\_id, p\_amount, v\_account\_id, p\_deposit\_condition, TRUNC(SYSDATE));  DBMS\_OUTPUT.PUT\_LINE('Successfully created deposit for client with id: ' || p\_user\_id);  DBMS\_OUTPUT.PUT\_LINE('Account for deposit is: ' || v\_account\_id);  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20002, 'No data found for the specified conditions');  WHEN VALUE\_ERROR THEN  RAISE\_APPLICATION\_ERROR(-20003, 'Invalid value passed');  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20004, 'An error occurred: ' || SQLERRM);  END CREATE\_DEPOSIT;  /  create procedure create\_deposit\_condition(p\_percentage in int, p\_name nvarchar2, p\_termin in int)  as  begin  if p\_percentage is null or p\_name is null or p\_termin is null then raise\_application\_error(-20001,'Conditions is incorrect'); end if;  if p\_termin>0 and p\_percentage>0 then  insert into DEPOSIT\_CONDITIONS(procentage\_per\_year, name, termin) values (p\_percentage,p\_name,p\_termin);  dbms\_output.put\_line('Deposit condition created successfully');  commit;  else raise\_application\_error(-20001,'Incorrect parametres');  end if;  end;  /  create PROCEDURE DELETE\_ACCOUNT\_FOR\_CLIENT(p\_client\_id IN INT, p\_account\_id IN INT) AS  v\_if\_exists INT;  BEGIN  -- Check if the account with the given ID and client ID exists  SELECT COUNT(\*)  INTO v\_if\_exists  FROM CLIENT\_ACCOUNT  WHERE ID = p\_account\_id  AND OWNER = p\_client\_id;  -- Handle the case when the account does not exist  IF v\_if\_exists = 0 THEN  -- Raise an exception for non-existent client or account  RAISE\_APPLICATION\_ERROR(-20001, 'Client with ID ' || p\_client\_id || ' or account with ID ' || p\_account\_id ||  ' does not exist');  end if;  -- Delete the account  select BALANCE into v\_if\_exists from CLIENT\_ACCOUNT where OWNER = p\_client\_id and ID = p\_account\_id;  if v\_if\_exists = 0 then  select IS\_LOCKED into v\_if\_exists from CLIENT\_ACCOUNT where OWNER = p\_client\_id and ID = p\_account\_id;  if v\_if\_exists = 0 then  DELETE  FROM CLIENT\_ACCOUNT  WHERE OWNER = p\_client\_id  AND ID = p\_account\_id;  DBMS\_OUTPUT.PUT\_LINE('Account deleted successfully');  else  raise\_application\_error(-20001, 'Given account is locked');  end if;  else  raise\_application\_error(-20001, 'Given account is not empty');  end if;  EXCEPTION  WHEN OTHERS THEN  -- Handle any other unexpected errors  RAISE\_APPLICATION\_ERROR(-20002, 'An error occurred: ' || SQLERRM);  END DELETE\_ACCOUNT\_FOR\_CLIENT;  /  create procedure delete\_deposit\_condition(p\_condition\_id in int)  as  v\_if\_exists int;  v\_if\_accounts int;  begin  if p\_condition\_id is null then raise\_application\_error(-20001, 'Condition id is null'); end if;  select count(\*) into v\_if\_exists from DEPOSIT\_CONDITIONS where ID = p\_condition\_id;  select count(\*) into v\_if\_accounts from DEPOSIT\_STATE inner join DEPOSIT\_CONDITIONS on DEPOSIT\_STATE.DEPOSIT\_TYPE = DEPOSIT\_CONDITIONS.ID where (extract(year from START\_DATE)+TERMIN)<extract(year from sysdate);  if v\_if\_exists > 0 and v\_if\_accounts=0 then  delete from DEPOSIT\_CONDITIONS where ID = p\_condition\_id;  DBMS\_OUTPUT.PUT\_LINE('Deposit condition deleted successfully');  commit;  else  raise\_application\_error(-20001, 'No deposit condition with such id or deposits with this condition didnt expired');  end if;  end delete\_deposit\_condition;  /  create PROCEDURE EXPORT\_CLIENT\_INFO AS  v\_json\_file UTL\_FILE.FILE\_TYPE;  v\_max\_id NUMBER;  p\_file\_path NVARCHAR2(100) := '/opt/oracle/oradata/ORCLCDB1/ORCLPDB1/your\_filename.json';  BEGIN  -- Open the file for writing  v\_json\_file := UTL\_FILE.FOPEN('EXPORT\_DIR', p\_file\_path, 'W');  -- Get the maximum ID  SELECT MAX(id) INTO v\_max\_id FROM CLIENT\_INFO;  -- Write the JSON array start  UTL\_FILE.PUT\_LINE(v\_json\_file, '[');  -- Cursor to fetch data from CLIENT\_INFO  FOR client\_rec IN (SELECT \* FROM CLIENT\_INFO)  LOOP  -- Write JSON object for each row  UTL\_FILE.PUT\_LINE(  v\_json\_file,  '{"name": "' || client\_rec.name || '"' ||  ',"surname": "' || client\_rec.surname || '"' ||  ',"thirdname": "' || client\_rec.thirdname || '"' ||  ',"birth\_date": "' || TO\_CHAR(client\_rec.birth\_date, 'YYYY-MM-DD') || '"' ||  ',"phone\_number": "' || client\_rec.phone\_number || '"}'  );  -- Write comma for all records except the last one  IF client\_rec.id != v\_max\_id THEN  UTL\_FILE.PUT\_LINE(v\_json\_file, ',');  END IF;  END LOOP;  -- Write the JSON array end  UTL\_FILE.PUT\_LINE(v\_json\_file, ']');  -- Close the file  UTL\_FILE.FCLOSE(v\_json\_file);  EXCEPTION  WHEN OTHERS THEN  -- Handle any other unexpected errors  DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);  END EXPORT\_CLIENT\_INFO;  /  create PROCEDURE generate\_account\_stats\_data  IS  v\_account\_id CLIENT\_ACCOUNT.id%TYPE;  v\_action\_date ACCOUNT\_STATS.action\_date%TYPE;  v\_action\_id ACTIONS.id%TYPE;  v\_amount ACCOUNT\_STATS.amount%TYPE;  BEGIN  FOR i IN 1..10000 LOOP  -- Генерация случайных данных  v\_account\_id := TRUNC(DBMS\_RANDOM.VALUE(145, 98124)); -- Замените границы в зависимости от ваших требований  v\_action\_date := TO\_DATE('01-JAN-2023', 'DD-MON-YYYY') + DBMS\_RANDOM.VALUE(1, 365); -- Замените границы в зависимости от ваших требований  v\_action\_id := TRUNC(DBMS\_RANDOM.VALUE(1, 4)); -- Значения id действий из таблицы ACTIONS (2, 3, 4)  v\_amount := DBMS\_RANDOM.VALUE(1, 1000); -- Замените границы в зависимости от ваших требований  -- Вставка данных в таблицу  INSERT INTO ACCOUNT\_STATS (account, action\_date, action, amount)  VALUES (v\_account\_id, v\_action\_date, v\_action\_id, v\_amount);  END LOOP;  COMMIT;  END generate\_account\_stats\_data;  /  create PROCEDURE READ\_JSON\_FILE AS  v\_bfile BFILE;  v\_json\_data CLOB;  p\_file\_path nvarchar2(60) := 'your\_filename.json';  BEGIN  -- Open the BFILE for reading  v\_bfile := BFILENAME('IMPORT\_DIR', '/opt/oracle/oradata/ORCLCDB1/ORCLPDB1/your\_filename.json');  DBMS\_LOB.FILEOPEN(v\_bfile, DBMS\_LOB.FILE\_READONLY);  -- Read the JSON data into a CLOB  DBMS\_LOB.CREATETEMPORARY(v\_json\_data, TRUE);  DBMS\_LOB.LOADFROMFILE(v\_json\_data, v\_bfile, DBMS\_LOB.GETLENGTH(v\_bfile));  -- Close the BFILE  DBMS\_LOB.FILECLOSE(v\_bfile);  -- Display JSON data on the console  DBMS\_OUTPUT.PUT\_LINE('Contents of the JSON File:');  DBMS\_OUTPUT.PUT\_LINE(v\_json\_data);  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('File not found: ' || p\_file\_path);  WHEN OTHERS THEN  -- Handle any other unexpected errors  DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);  END READ\_JSON\_FILE;  /  create PROCEDURE generate\_client\_info\_data  IS  v\_name CLIENT\_INFO.name%TYPE;  v\_surname CLIENT\_INFO.surname%TYPE;  v\_thirdname CLIENT\_INFO.thirdname%TYPE;  v\_birth\_date CLIENT\_INFO.birth\_date%TYPE;  v\_phone\_number CLIENT\_INFO.phone\_number%TYPE;  BEGIN  FOR i IN 1..100000  LOOP  -- Генерация случайных данных  v\_name := DBMS\_RANDOM.STRING('A', 10); -- Замените 10 на максимальную длину имени  v\_surname := DBMS\_RANDOM.STRING('A', 10); -- Замените 10 на максимальную длину фамилии  v\_thirdname := DBMS\_RANDOM.STRING('A', 10); -- Замените 10 на максимальную длину отчества  v\_birth\_date := TO\_DATE('01-JAN-1970', 'DD-MON-YYYY') +  DBMS\_RANDOM.VALUE(1, 200000); -- Замените границы в зависимости от ваших требований  v\_phone\_number := TO\_CHAR(TRUNC(DBMS\_RANDOM.VALUE(1000000000, 9999999999)));  -- Генерация случайного 10-значного номера  -- Вставка данных в таблицу  INSERT INTO CLIENT\_INFO (name, surname, thirdname, birth\_date, phone\_number)  VALUES (v\_name, v\_surname, v\_thirdname, v\_birth\_date, v\_phone\_number);  END LOOP;  COMMIT;  END generate\_client\_info\_data;  /  create PROCEDURE generate\_random\_data  IS  v\_owner\_id int;  v\_is\_locked NUMBER(1);  v\_account\_type NUMBER(1);  v\_balance NUMBER;  BEGIN  FOR i IN 1..100000  LOOP  -- Генерация случайных данных  v\_owner\_id := TRUNC(DBMS\_RANDOM.VALUE(145, 98124)); -- Замените границы в зависимости от ваших требований  v\_is\_locked := TRUNC(DBMS\_RANDOM.VALUE(0, 2));  v\_account\_type := TRUNC(DBMS\_RANDOM.VALUE(0, 2));  v\_balance :=trunc(DBMS\_RANDOM.VALUE(0, 100000),2);  -- Замените границы в зависимости от ваших требований  -- Вставка данных в таблицу  INSERT INTO CLIENT\_ACCOUNT (owner, is\_locked, account\_type, balance)  VALUES (v\_owner\_id, v\_is\_locked, v\_account\_type, v\_balance);  END LOOP;  COMMIT;  END generate\_random\_data;  /  CREATE OR REPLACE TRIGGER after\_insert\_deposit\_conditions  after INSERT ON DEPOSIT\_CONDITIONS  FOR EACH ROW  DECLARE  BEGIN  -- You can add additional conditions or logic here if needed  -- Print a message to DBMS\_OUTPUT  DBMS\_OUTPUT.PUT\_LINE('Deposit condition created');  END after\_insert\_deposit\_conditions;  create or replace directory JSONDIR as '/opt/oracle/oradata/ORCLCDB1/ORCLPDB1';  CREATE OR REPLACE PROCEDURE read\_json\_file(P\_FILE\_NAME NVARCHAR2) AS  v\_clob\_data CLOB;  v\_file\_handle UTL\_FILE.FILE\_TYPE;  BEGIN  -- Открытие файла для чтения  v\_file\_handle := UTL\_FILE.FOPEN('JSONDIR', P\_FILE\_NAME, 'r');  BEGIN  LOOP  -- Чтение данных из файла  UTL\_FILE.GET\_LINE(v\_file\_handle, v\_clob\_data);  EXIT WHEN v\_clob\_data IS NULL;  -- Вывод JSON-данных в консоль  DBMS\_OUTPUT.PUT\_LINE(v\_clob\_data);  END LOOP;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  NULL;  END;  -- Закрытие файла  UTL\_FILE.FCLOSE(v\_file\_handle);  -- Вывод завершающего сообщения  DBMS\_OUTPUT.PUT\_LINE('Чтение JSON-данных завершено');  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Ошибка при чтении JSON-данных: ' || SQLERRM);  END read\_json\_file;  / |

Приложение С Создание триггера

|  |
| --- |
| CREATE OR REPLACE TRIGGER after\_insert\_deposit\_conditions  after INSERT ON DEPOSIT\_CONDITIONS  FOR EACH ROW  DECLARE  BEGIN  -- You can add additional conditions or logic here if needed  -- Print a message to DBMS\_OUTPUT  DBMS\_OUTPUT.PUT\_LINE('Deposit condition created');  END after\_insert\_deposit\_conditions; |