

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 предприятия «Лидское пиво» с реализацией
технологии шифрования и маскирования в БД»

Выполнил студент Викторович И.С.
(Ф.И.О.)
Руководитель работы 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	5
1 Постановка задачи	6
2 Проектирование базы данных	7
2.1 Схема базы данных	7
2.2 Таблицы базы данных	8
2.3 Вывод по разделу	10
3 Разработка объектов базы данных	11
3.1 Табличное пространство	11
3.1 Таблицы	11
3.2 Процедуры	11
3.3 Функции	12
3.4 Вывод по разделу	12
4 Описание процедур экспорта и импорта	13
4.1 Описание процедуры экспорта	13
4.2 Описание процедуры импорта	13
4.3 Вывод по разделу	14
5 Тестирование производительности	15
5.1 Тестирование запросов заполнения таблиц	15
5.2 Проверка технологии шифрования и маскирования	16
5.3 Вывод по разделу	17
6 Описание и применение технологии в БД	18
6.1 Описание взаимодействия с технологией	18
6.2 Вывод по разделу	21
7 Руководство пользователя	22
Вход в систему: Для доступа к базе данных используйте ваше уникальное имя пользователя и пароль. Проверьте, что ваши учетные данные корректны.	22
Заключение	23
Список используемых источников	24
Приложение А Создание таблиц	25
Приложение Б Создание функций	27
Приложение С Процедуры	28

Введение

Современные предприятия, в том числе и компания «Лидское пиво», сталкиваются с необходимостью эффективного управления своими ресурсами, оптимизации бизнес-процессов и обеспечения высокого уровня обслуживания клиентов. В этом контексте разработка базы данных, специализированной для учета товаров, заказов и складских операций, играет ключевую роль.

Целью данного проекта является создание надежной и функциональной базы данных, которая позволит предприятию «Лидское пиво» решать следующие задачи:

1 Учет товаров и складских операций: База данных будет хранить информацию о наличии товаров на складе, движении товаров (поступление, списание, перемещение) и текущем состоянии запасов. Это позволит компании точно контролировать свои ресурсы и избегать дублирования данных.

2 Регистрация заказов и обработка клиентских запросов: Система будет отслеживать заказы клиентов, обрабатывать их быстро и эффективно, а также предоставлять актуальную информацию о статусе заказа. Это поможет удовлетворить потребности клиентов, минимизировать время доставки и повысить уровень обслуживания.

3 Анализ данных и принятие обоснованных решений: База данных будет предоставлять аналитические данные о реализации продукции, популярности определенных товаров, спросе на них и других ключевых показателях. Это поможет руководству компании принимать обоснованные решения о закупках, управлении запасами и стратегии развития.

4 Безопасность данных: Важным аспектом разработки будет обеспечение безопасности данных. Применение технологий шифрования и маскирования поможет защитить конфиденциальные сведения о клиентах, поставщиках и других участниках бизнес-процессов.

В итоге, использование разработанной базы данных способствует повышению конкурентоспособности компании, удовлетворению потребностей клиентов и эффективному управлению ресурсами.

1 Постановка задачи

В рамках данного проекта требуется спроектировать инфраструктуру базы данных предприятия «Лидское пиво». 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 и написание SQL-скриптов.

Для заполнения таблиц данными предусмотрен импорт из JSON-файлов. Кроме того, будет использована технология Oracle шифрования и маскирования для хеширования паролей пользователей.

Проект был разработан в Oracle с использованием sqldeveloper, где реализован функционал и возможности, предоставляемые базой данных через процедуры и функции. Важными функциями приложения будут:

- управление заказами (добавление, удаление, изменение);
- определение ролей (администратор, пользователь);
- управление продуктами (добавление, удаление, изменение);
- анализ продукции (количество проданных товаров по периодам, популярные товары, общее количество продукции);

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.

Применение технологии шифрования и маскирования для защиты пользовательских данных в базе данных позволит проанализировать преимущества, которые предоставляет данная технология.

2 Проектирование базы данных

2.1 Схема базы данных

Для курсового проекта используется PDB с настроенными параметрами. Определены таблицы, необходимые для хранения информации, используемой реализации требуемого функционала.

В базе данных предприятия пивоваренного завода применяются таблицы для хранения информации о доступной продукции: сорта пива, кваса и других напитков, пивоваренных заводах, складах предприятия, закупках завода, пользовательских данных, а также пользовательских заказов. Диаграмма таблиц и их отношения представлены на рисунке 2.1.

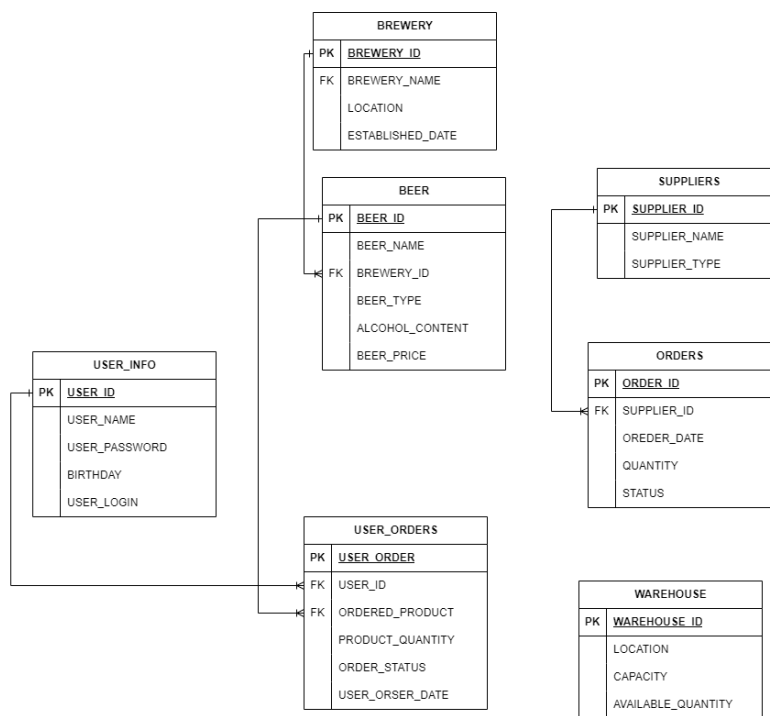


Рисунок 2.1 – Схема базы данных

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 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, процедурам и данным, соответствующим его роли.

Взаимодействие с системой наглядно демонстрирует Use-case диаграмма – рисунок 2.2.

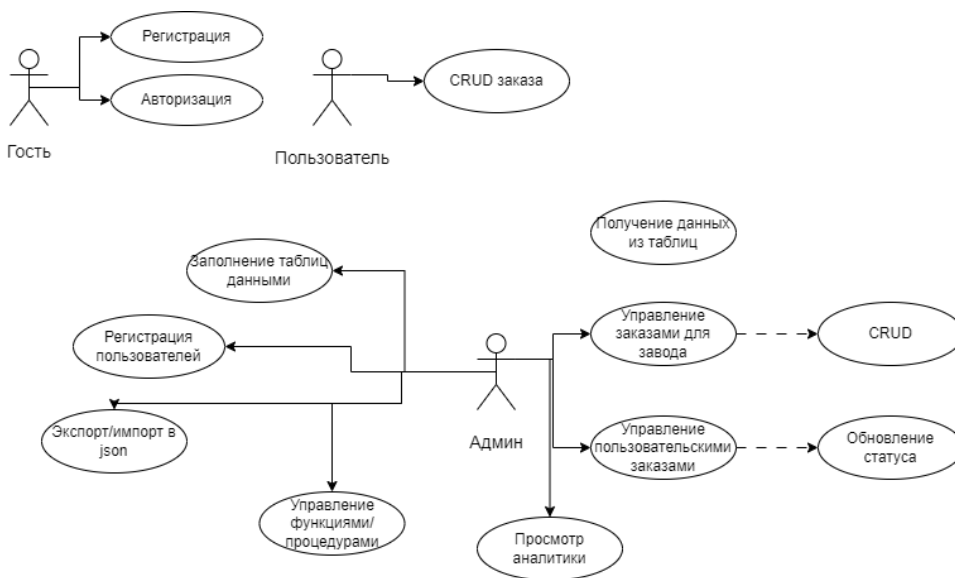


Рисунок 2.2 – Use Case диаграмма

В проекте реализован функционал для трёх групп пользователей – Администраторы, зарегистрированные пользователи и гости. Все действия данных пользователей будут выполняться через хранимые процедуры и функции.

2.2 Таблицы базы данных

Далее приведено описание таблиц базы данных: названия, названия столбцов, типы данных столбцов, описание содержания столбцов. Описание представлено в таблицах 2.1 – 2.10.

Таблица 2.1 – Содержание таблицы BREWERY (Пивоварни)

Название	Тип данных	Описание
Brewery_id	NUMBER	Уникальный идентификатор пивоварни
Brewery_name	VARCHAR2(100)	Название пивоварни
Location	VARCHAR2(100)	Местонахождение пивоварни
Established_Date	DATE	Дата основания

Добавлено примечание ([An1]): А где 2.1? 2.2?

Нумерация рисунков, таблиц и листингов идет отдельно

Эта таблица хранит в себе данные пивоварнях, относящихся к предприятию.

Таблица 2.2 – Содержание таблицы BEER (Продукция)

Название	Тип данных	Описание
Beer_id	NUMBER	Уникальный идентификатор пива
Beer_name	VARCHAR2(100)	Название пива/напитка
Brewery_id	NUMBER	Уникальный идентификатор пивоварни

Продолжение таблицы 2.2

Название	Тип данных	Описание
Beer_type	VARCHAR2(50)	Тип пива/напитка
Alcohol_content	NUMBER	Процент алкоголя в продукте
Beer_price	NUMBER	Стоимость пива

В этой таблице содержатся данные продукции, производимой на заводе «Лидское пиво».

Таблица 2.3 – Содержание таблицы Suppliers (Поставщики)

Название	Тип данных	Описание
Supplier_id	NUMBER	Уникальный идентификатор поставщика
Supplier_name	VARCHAR2(100)	Название поставщика
Supplier_type	VARCHAR2(50)	Тип поставщика(по сырью)

Эта таблица хранит в себе данные о поставщиках.

Таблица 2.4 – Содержание таблицы Orders (Заказы предприятия)

Название	Тип данных	Описание
Order_id	NUMBER	Уникальный идентификатор заказа
Supplier_id	NUMBER	Идентификатор поставщика
Order_date	DATE	Дата заказа
Quantity	NUMBER	Количество единиц в заказе
Status	VARCHAR2(20)	Статус выполнения заказа

В данной таблице содержатся данные закупках предприятия.

Таблица 2.5 – Содержание таблицы Warehouse (склады)

Название	Тип данных	Описание
Warehouse_ID	NUMBER	Уникальный идентификатор склада
Location	VARCHAR2(100)	Месторасположение склада
Capacity	NUMBER	Общая вместимость склада
Avaiable_capacity	NUMBER	Доступная вместимость склада

В данной таблице содержится информация о складах завода.

Таблица 2.6 – Содержание таблицы User_info (Пользовательская информация)

Название	Тип данных	Описание
----------	------------	----------

User_id	NUMBER	Уникальный идентификатор пользователя
User_name	VARCHAR2(40)	Имя пользователя
User_password	VARCHAR2(200)	Хешированный пароль пользователя
Birthday	DATE	Дата рождения пользователя
User_login	VARCHAR2(20)	Логин пользователя

В данной таблице содержится регистрационная информация о пользователе.

Таблица 2.7 – Содержание таблицы User_orders (Пользовательские заказы)

Название	Тип данных	Описание
Users_order_id	NUMBER	Уникальный идентификатор заказа

Продолжение таблицы 2.7

Название	Тип данных	Описание
User_id	NUMBER	Идентификатор пользователя
Ordered_product	NUMBER	Наименования продукта в заказе
Product_quantity	NUMBER	Количество продуктов в заказе
Order_status	VARCHAR2(20)	Статус заказа
Order_date	DATE	Дата создания заказа

В данной таблице содержится информация о заказах пользователей.

2.3 Вывод по разделу

В данном разделе создана подключаемая база данных, спроектирована структура таблиц базы данных, реализованы таблицы, столбцы и типы данных для них, определены сценарии взаимодействия пользователей базы данных завода с предоставляемыми данными, определены связи между таблицами, а также логика взаимодействия с ними.

3 Разработка объектов базы данных

3.1 Табличное пространство

Для каждой таблицы и пользователя создаётся табличное пространство. Листинг показывает пример создания такого табличного пространства для таблицы Beer – листинг 3.1

```
CREATE TABLESPACE beer_space  
DATAFILE 'beer_tspace.dbf' SIZE 100M  
AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED  
EXTENT MANAGEMENT LOCAL;
```

Листинг 3.1 – Создание табличного пространства

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

3.1 Таблицы

Можно отметить, что в каждой таблице есть поле, представляющее ID, которое является первичным ключом и заполняется с помощью последовательности с шагом 1 либо присвоением уникального идентификатора при заполнении таблицы данными. Числовые типы данных представлены типами NUMBER. Используется тип DATE для хранения даты, например, даты оформления заказа, даты рождения. Символьные данные представлены типами varchar2(n), где n – максимальная длина последовательности символов.

После создания таблицы заполняются некоторым количеством тестовых данных для проверки работоспособности и разработки объектов следующих пунктов.

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

3.2 Процедуры

В данном проекте широко представлено применение процедур. С помощью процедур реализованы такие операции, как добавление, удаление и обновление данных, регистрация пользователей, исполнение технологии шифрования и маскирования в базе данных, аналитика продукции.

Администратор имеет привилегии dba, следовательно, имеет право выполнить практически любую процедуру. Для пользователей администратором специально выдаются права на исполнение конкретных процедур, определённых их функционалом.

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

3.3 Функции

В данном курсовом проекте функции применены при аутентификации пользователей. Это обусловлено тем, что эта функция многократно вызывается в различных процедурах для выполнения определённых действий.

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

3.4 Вывод по разделу

В данном разделе было описано и выполнено создание объектов для базы данных предприятия, таких как табличные пространства, роли и пользователи, таблицы, хранимые процедуры, функции и триггеры. Определено взаимодействие представленных объектов базы данных, добавлены обработчики исключений, выданы привилегии на вызов и использование пользователями процедур, функций и других объектов.

4 Описание процедур экспорта и импорта

4.1 Описание процедуры экспорта

Для хранения данных используется JSON-формат. Процедура экспорта в JSON-формат для продукции завода – листинг 4.1.

```
CREATE OR REPLACE DIRECTORY UTL_DIR AS '/home/oracle/JSON';
GRANT READ, WRITE ON DIRECTORY UTL_DIR TO public;
CREATE OR REPLACE DIRECTORY UTL_DIR AS
'C:\WINDOWS.X64_193000_db_home\database';

CREATE OR REPLACE PROCEDURE ExportToJson IS
  v_file UTL_FILE.FILE_TYPE;
  v_row BEER%ROWTYPE;
BEGIN
  v_file := UTL_FILE.FOPEN('UTL_DIR', 'FACTORY_PRODUCTS.json', 'W');
  UTL_FILE.PUT_LINE(v_file, '[');

  FOR v_row IN (SELECT JSON_OBJECT(
    'beer_id' is CAST(BEER_ID as NUMBER),
    'beer_name' is CAST(BEER_NAME as VARCHAR(100)),
    'brewery_id' is CAST(BREWERY_ID as NUMBER),
    'beer_type' is CAST(BEER_TYPE as VARCHAR(50)),
    'alcohol_content' is CAST(ALCOHOL_CONTENT as NUMBER),
    'beer_price' is CAST(BEER_PRICE as NUMBER)
  ) AS JSON_DATA
  FROM BEER)
  LOOP
    UTL_FILE.PUT_LINE(v_file, v_row.JSON_DATA || ',');
  END LOOP;

  UTL_FILE.PUT_LINE(v_file, ']');
  UTL_FILE.FCLOSE(v_file);
EXCEPTION
WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);
RAISE;
END;
```

Листинг 4.1 – Процедура для экспорта данных из таблицы Beer

Процедура выполняет экспорт данных из таблицы Beer в формат JSON. Результат экспорта данных сохраняется в файл 'FACTORY_PRODUCTS.json'. В данной процедуре применяется UTL FILE для работы с файлами, создания JSON-объектов для каждой строки таблицы, которые записываются в виде JSON-массива.

4.2 Описание процедуры импорта

Реализована процедура для импорта данных из формата JSON. Данная процедура представлена в листинге 4.2.

```

CREATE OR REPLACE PROCEDURE ImportFromJSON
IS
BEGIN
FOR json_rec IN (
    SELECT beer_id, beer_name, brewery_id, beer_type, alcohol_content,
beer_price
FROM JSON_TABLE(BFILENAME('UTL_DIR', 'FACTORY_PRODUCTS.json'),
'$[*]' COLUMNS (
    beer_id number PATH '$.beer_id',
    beer_name varchar2(100) PATH '$.beer_name',
    brewery_id number PATH '$.brewery_id',
    beer_type varchar2(50) PATH '$.beer_type',
    alcohol_content number PATH '$.alcohol_content',
    beer_price number PATH '$.beer_price'
))
)
LOOP
BEGIN
    INSERT INTO JSONBEER (beer_id, beer_name, brewery_id, beer_type,
alcohol_content, beer_price )
    VALUES (json_rec.beer_id, json_rec.beer_name,
json_rec.brewery_id, json_rec.beer_type,
json_rec.alcohol_content, json_rec.beer_price);
    DBMS_OUTPUT.PUT_LINE('Product was added successfully');
EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Product with the id already exists. ');
WHEN OTHERS THEN
ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Error inserting user: ' || SQLERRM);
RAISE;
END;
END LOOP;
END;

```

Листинг 4.2 – Процедура для импорта данных в таблицу JSONBEER

Процедура ImportFromJSON выполняет импорт данных из JSON-формата в специально созданную таблицу JSONBEER для демонстрации корректной работы процедуры. Реализован механизм обработки исключений. В результате импорта мы получим заполненную, ранее экспортированными, данными таблицу JSONBEER.

4.3 Вывод по разделу

В этом разделе были рассмотрены процедуры, выполняющие экспорт/импорт данных в/из формата JSON. Данный механизм позволяет расширить функциональные возможности базы данных, позволяет оптимизировать ввод данных.

5 Тестирование производительности

5.1 Тестирование запросов заполнения таблиц

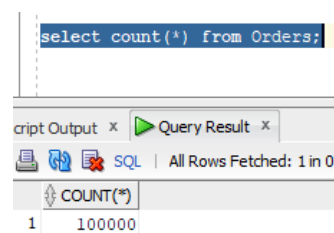
В базе данных реализованы различные процедуры, функции, а также таблицы заполнены необходимым количеством данных – листинг 5.1. Теперь необходимо протестировать, что всё работает корректно.

```

DECLARE
    v_supplier_id NUMBER;
    v_order_date DATE;
    v_quantity NUMBER;
    v_status VARCHAR2(20);
BEGIN
    FOR i IN 1..100000 LOOP
        v_supplier_id := TRUNC(DBMS_RANDOM.VALUE(1, 20));
        v_order_date := SYSDATE - TRUNC(DBMS_RANDOM.VALUE(1, 365));
        v_quantity := TRUNC(DBMS_RANDOM.VALUE(1, 1000));
        v_status := CASE
            WHEN i < 30000 THEN 'is complited'
            WHEN i < 60000 THEN 'is compliting'
            ELSE 'rejected' END;
        AddOrder(v_supplier_id, TO_CHAR(v_order_date, 'DD.MM.YYYY'),
            v_quantity, v_status);
    END LOOP;
END;
```

Листинг 5.1 – Заполнение таблицы Orders

При успешном выполнении скрипта, представленного на листинге 5.1, таблица Orders заполняется 100000 строк. Убедится в этом можно, выполнив запрос `SELECT count(*) FROM ORDERS;` Результат выполнения запроса представлен на рисунке 5.1



The screenshot shows a SQL query execution window with the query `select count(*) from Orders;` entered in the top text area. Below the query, there are tabs for 'Script Output' and 'Query Result'. The 'Query Result' tab is active, displaying the result of the query. The status bar indicates 'All Rows Fetched: 1 in 0'. The result is shown in a table with one row and one column labeled 'COUNT(*)', containing the value '100000'.

COUNT(*)
100000

Рисунок 5.1 – Результат выполнения запроса



```

DECLARE
v_supplier_id NUMBER;
v_order_date DATE;
v_quantity NUMBER;
v_status VARCHAR2(20);
BEGIN
FOR i IN 1..100000 LOOP
v_supplier_id := TRUNC(DEMS_RANDOM.VALUE(1, 20)); -- 20 поставщиков
v_order_date := SYSDATE - TRUNC(DEMS_RANDOM.VALUE(1, 365)); -- случайные даты за последний год
v_quantity := TRUNC(DEMS_RANDOM.VALUE(1, 1000)); -- случайные количества

v_status := CASE
WHEN i < 30000 THEN 'is complited' -- первые 30 000 заказов выполнены
WHEN i < 60000 THEN 'is compliting' -- следующие 30 000 заказов в процессе
ELSE 'rejected' -- остальные отклонены
END;
AddOrder(v_supplier_id, TO_CHAR(v_order_date, 'DD.MM.YYYY'), v_quantity, v_status);
END LOOP;
END;
--ALTER SYSTEM FLUSH BUFFER_CACHE;

```

Script Output x Query Result x

Task completed in 17,825 seconds

100 000 rows deleted.

Рисунок 5.2 – Время поиска пользователя

Из рисунка 5.2 можно заметить, что время выполнения скрипта составило 17,825 секунд. На время выполнения запроса могут влиять многие факторы, такие как количество добавляемых данных или загруженность системы.

5.2 Проверка технологии шифрования и маскирования

Одной из задач курсового проекта была реализация технологии шифрования и маскирования. Следует выполнить тестирование процедур, реализующих эту технологию, для проверки корректной работы.

Листинги процедур, реализующих технологию находятся в приложении Г.

В данном курсовом проекте шифрование используется для скрытия паролей для пользователей базы данных методом хеширования паролей.

Для тестирования выполним нового пользователя, а затем выполним select-запрос к таблице User_info, где, согласно коду процедуры, вместо истинного пароля должен находится его хеш.

```

DECLARE
result_message VARCHAR2(100);
BEGIN
c##admin_user.RegisterUser('Liza', '204', '204', '15-07-2005', 'lizaspostav', result_message);
DEMS_OUTPUT.PUT_LINE(result_message);
END;

```

Рисунок 5.3 – Вызов процедуры регистрации

```
select * from user_info;
```

Script Output x Query Result x

SQL | All Rows Fetched: 5 in 0,002 seconds

USER_ID	USER_NAME	USER_PASSWORD	BIRTHDAY	USER_LOGIN
1	Iryna	FC56DBC6D4652B315B86B71C8D688C1CCDEA9C5F1FD07763D2659FDE2E2FC49A	13.03.04	leviathan
2	Ilya	FC56DBC6D4652B315B86B71C8D688C1CCDEA9C5F1FD07763D2659FDE2E2FC49A	21.03.03	svendell
3	21 Anna	FC56DBC6D4652B315B86B71C8D688C1CCDEA9C5F1FD07763D2659FDE2E2FC49A	21.03.03	anklko
4	22 Nikita	FC56DBC6D4652B315B86B71C8D688C1CCDEA9C5F1FD07763D2659FDE2E2FC49A	28.03.04	hamster
5	24 Liza	FC56DBC6D4652B315B86B71C8D688C1CCDEA9C5F1FD07763D2659FDE2E2FC49A	15.07.05	lizaspostav

Рисунок 5.4 – Запрос к таблице User_info

Из результирующего набора мы видим, что шифрование данных выполняется корректно, истинные пароли скрываются, вместо него – хеш.

5.3 Вывод по разделу

Выполнено тестирование заполнения таблицы Orders 100000 строками данных, проверено время выполнения, рассмотрены причины полученных результатов. Протестированы процедуры, отвечающие за реализацию технологий шифрования и маскирования.

6 Описание и применение технологии в БД

6.1 Описание взаимодействия с технологией

Технология маскирования данных в базе данных - это метод обеспечения безопасности информации путем скрытия или искажения конфиденциальных данных от пользователей, которым нет необходимости видеть их полностью.

В курсовом проекте данная технология реализована в виде представления, выполняющего select-запрос. В перечислении столбцов таблицы после ключевого слова select некоторые данные, требующие маскирования, реализовывались через case, где, согласно реализации, для администратора данные остаются в изначальном виде, а для остальных пользователей происходит замена данных конкретного столбца на условные символы.

Убедится в корректной работе технологии маскирования можно выполнив select-запрос, вызывающий представление Masked_User_info.

```
CREATE OR REPLACE VIEW Masked_User_info AS
SELECT
  user_id,
  CASE
    WHEN SYS_CONTEXT('USERENV', 'CURRENT_SCHEMA') = 'C##ADMIN_USER' THEN
      user_name
    ELSE 'xxxxxxx'
  END AS user_name,
  CASE
    WHEN SYS_CONTEXT('USERENV', 'CURRENT_SCHEMA') = 'C##ADMIN_USER' THEN
      user_password
    ELSE '*****'
  END AS user_password,
  birthday,
  user_login
FROM User_info;
```

Листинг 6.1 – Представление с реализацией технологии маскирования

The screenshot shows a SQL query editor with the following SQL code:

```

select * from user_info;

CREATE OR REPLACE VIEW Masked_User_info AS
SELECT
  user_id,
  CASE
    WHEN SYS_CONTEXT('USERENV', 'CURRENT_SCHEMA') = 'C##ADMIN_USER' THEN user_name
    ELSE 'xxxxxxx'
  END AS user_name,
  CASE
    WHEN SYS_CONTEXT('USERENV', 'CURRENT_SCHEMA') = 'C##ADMIN_USER' THEN user_password
    ELSE 'xxxxxxx'
  END AS user_password,
  birthday,
  user_login
FROM User_info;

select * from c##admin_user.Masked_User_info;

grant select on Masked_User_info to c##app_role;

```

The bottom panel shows the query results for the last query:

USER_ID	USER_NAME	USER_PASSWORD	BIRTHDAY	USER_LOGIN
1	Iryna	FC56DBC6D4652B315B86B71C8D688C1CCDEA9C5F1FD07763D2659FDE2E2FC49A	13.03.04	leviathan
2	Ilya	FC56DBC6D4652B315B86B71C8D688C1CCDEA9C5F1FD07763D2659FDE2E2FC49A	21.03.03	svendell
3	Anna	FC56DBC6D4652B315B86B71C8D688C1CCDEA9C5F1FD07763D2659FDE2E2FC49A	21.03.03	anklko
4	Nikita	FC56DBC6D4652B315B86B71C8D688C1CCDEA9C5F1FD07763D2659FDE2E2FC49A	28.03.04	hamster
5	Liza	FC56DBC6D4652B315B86B71C8D688C1CCDEA9C5F1FD07763D2659FDE2E2FC49A	15.07.05	lizaspostav
6	Sasha	FC56DBC6D4652B315B86B71C8D688C1CCDEA9C5F1FD07763D2659FDE2E2FC49A	13.04.04	miltoha

Рисунок 6.1 – Вызов представления администратором

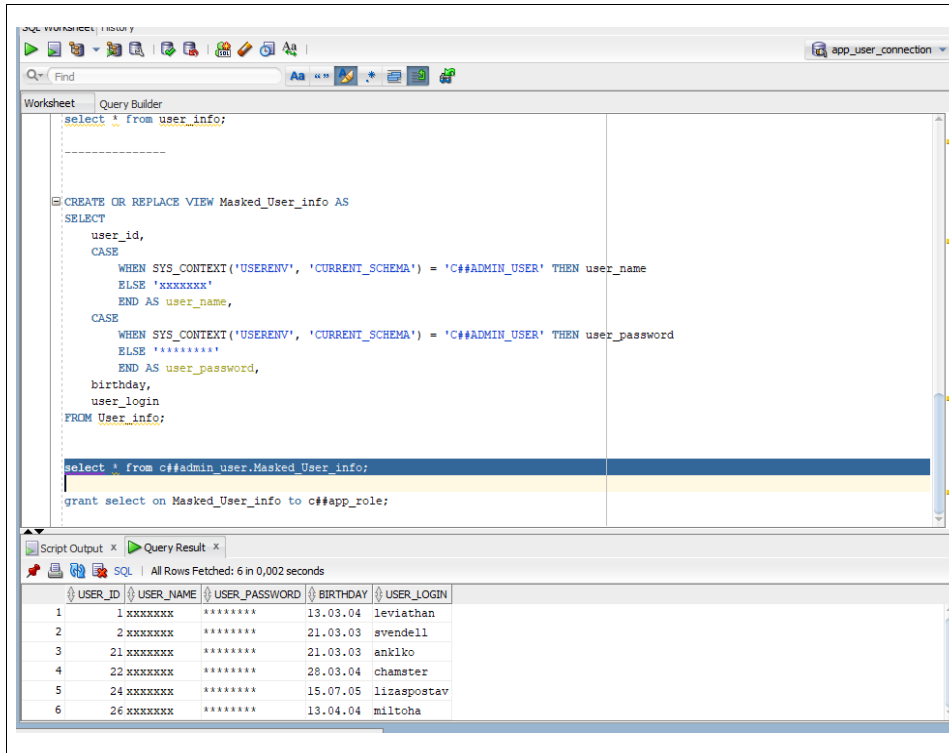


Рисунок 6.2 – Вызов представления пользователем

Один из методов шифрования данных в Oracle – хеширование. В Oracle используют функции хеширования, такие как *DBMS_CRYPTO.HASH*.

В данном курсовом проекте шифрование используется при регистрации пользователей для хеширования сохранённых паролей.

В процедуре, реализующей регистрацию пользователей, применяются функция хеширования, описанная выше. В таблицу записывается хеш. В нашей работе функция *DBMS_CRYPTO.HASH* принимает в качестве входных данных байтовое представление пароля и использует алгоритм SHA-256 для генерации хеша.

Данная операция является необратимой, то есть нельзя преобразовать хеш к паролю. Процесс проверки корректности следующий:

1. пользователь вводит свой пароль для аутентификации;
2. введённый пароль хешируется с помощью того же алгоритма, что и при сохранении в базе данных;
3. сгенерированный хеш ввода сравнивается с хешем пароля, сохранённым в базе данных;
4. если хеши совпадают, то совпадают и пароли.

6.2 Вывод по разделу

Технологии шифрования и маскирования данных - это важные методы обеспечения безопасности и конфиденциальности информации в базах данных.

Шифрование данных позволяет преобразовывать информацию в зашифрованный формат, который невозможно прочесть без ключа шифрования. Это способ защиты информации в базе данных от несанкционированного доступа к конфиденциальным данным, даже если злоумышленники получили физический доступ к базе данных.

Маскирование данных, с другой стороны, позволяет скрыть чувствительные данные для предотвращения их отображения или раскрытия пользователям или приложениям, которые не имеют соответствующих прав доступа. Это полезный метод для обеспечения конфиденциальности данных в различных сценариях, например, при отображении частичной информации для пользователей, которые не имеют привилегий для полного доступа к данным.

Оба подхода эффективны в защите информации в базах данных, помогая предотвратить утечку конфиденциальной информации и обеспечивая соблюдение требований безопасности и законодательства о защите данных. Однако важно правильно выбирать и реализовывать эти методы, учитывая специфику и требования вашего проекта или организации.

7 Руководство пользователя

Вход в систему: Для доступа к базе данных используйте ваше уникальное имя пользователя и пароль. Проверьте, что ваши учетные данные корректны.

Функции базы данных:

1. Управление продукцией:

- **Добавление новых продуктов:** Позволяет добавлять новые виды пива в систему, указывая их характеристики и цены.
- **Изменение данных о продукции:** Редактирование информации о продукции (цена, объемы производства и т.д.).
- **Управление складом:** Отслеживание количества продукции на складе.

2. Заказы:

- **Размещение заказов:** Возможность размещения заказов на поставку продукции.
- **Отслеживание статуса заказов:** Мониторинг текущего состояния размещенных заказов.

3. Аналитика:

- **Отчеты и анализ данных:** Генерация отчетов о продажах, объемах производства и других аналитических данных.
- **Статистика производства:** Просмотр статистики производства для оптимизации работы завода.

4. Управление пользователями:

- **Добавление и удаление пользователей:** Управление учетными записями пользователей с различными уровнями доступа.

Заключение

В ходе выполнения курсового проекта была спроектирована и реализована инфраструктура базы данных для предприятия «Лидское пиво», которая содержит информацию о пользователях, продуктах и заказах, с использованием СУБД Oracle 12c. 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.

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 предприятия «Лидское пиво» с широким функционалом, позволяющим эффективно управлять продуктами, заказами и пользователями. Проект демонстрирует преимущества использования СУБД Oracle 12c для создания мощной и гибкой инфраструктуры базы данных.

Добавлено примечание ([CM2]): Добавить про шифрование и маски

Список используемых источников

1. Нистюк О. А. Курс лекций по базам данных, [Электронный ресурс]. – Режим доступа: <https://diskstation.belstu.by:5001/>. – Дата доступа: 01.10.2023.
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: 01.10.2023.
3. Конфиденциальность и функции защиты данных, [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/cloud/latest/related-docs/OMCEZ/ru/EncryptionAtRest.htm>. – Дата доступа: 01.11.2023

Приложение А Создание таблиц

```
CREATE TABLE Brewery (
    Brewery_ID NUMBER PRIMARY KEY,
    Brewery_Name VARCHAR2(100),
    Location VARCHAR2(100),
    Established_Date DATE
)
TABLESPACE brewery_space;

CREATE TABLE Beer (
    Beer_ID NUMBER PRIMARY KEY,
    Beer_Name VARCHAR2(100),
    Brewery_ID NUMBER REFERENCES Brewery(Brewery_ID),
    Beer_Type VARCHAR2(50),
    Alcohol_Content NUMBER,
    Beer_price NUMBER
)
TABLESPACE beer_space;

CREATE TABLE Suppliers (
    Supplier_ID NUMBER PRIMARY KEY,
    Supplier_Name VARCHAR2(100),
    Supplier_Type VARCHAR2(50)
)
TABLESPACE supplier_space;

CREATE TABLE Orders (
    Order_ID NUMBER PRIMARY KEY,
    Supplier_ID NUMBER REFERENCES Suppliers(Supplier_ID),
    Order_date DATE,
    Quantity NUMBER,
    Status VARCHAR2(20),
    CONSTRAINT check_status check (Status in ('is complited', 'is
compliting', 'rejected'))
) TABLESPACE orders_space;

CREATE TABLE Warehouse (
    Warehouse_ID NUMBER PRIMARY KEY,
    Location VARCHAR2(100),
    Capacity NUMBER,
    Available_quantity NUMBER
) TABLESPACE warehouse_space;

create table User_info (
    user_id NUMBER PRIMARY KEY,
    user_name varchar2(40),
    user_password varchar(200),
    birthday DATE,
    user_login varchar(20)
) TABLESPACE user_space;
```

```
create table User_orders (  
    users_order_id NUMBER PRIMARY KEY,  
    user_id NUMBER REFERENCES User_info(user_id),  
    ordered_product NUMBER REFERENCES Beer(Beer_ID),  
    product_quantity NUMBER  
) TABLESPACE user_orders;  
  
ALTER TABLE User_orders  
ADD order_status VARCHAR2(20) DEFAULT 'active' check (order_status  
in ('completed', 'active', 'rejected'));  
alter table User_orders  
add user_order_date DATE;
```


Приложение Б Создание функций

```
CREATE OR REPLACE FUNCTION AuthenticateUser(
    p_user_login IN VARCHAR2,
    p_user_password IN VARCHAR2
) RETURN BOOLEAN
IS
    hashed_password RAW(256);
    stored_password RAW(256);
BEGIN
    SELECT user_password INTO stored_password FROM User_info WHERE
    user_login = p_user_login;

    hashed_password :=
    DBMS_CRYPTO.HASH(UTL_RAW.CAST_TO_RAW(p_user_password),
    DBMS_CRYPTO.HASH_SH256);

    DBMS_OUTPUT.PUT_LINE('Comparing stored password: ' ||
    stored_password || ' with hashed password: ' || hashed_password);

    IF hashed_password = stored_password THEN
    DBMS_OUTPUT.PUT_LINE('User authenticated successfully for login: '
    || p_user_login);
    RETURN TRUE;
    ELSE
    DBMS_OUTPUT.PUT_LINE('Authentication failed for login: ' ||
    p_user_login);
    RETURN FALSE;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('User not found for login: ' || p_user_login);
    RETURN FALSE;
    WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred during authentication for
    login: ' || p_user_login);
    RETURN FALSE;
END;
```

Приложение С Процедуры

```

CREATE OR REPLACE PROCEDURE AddOrder (
    Supplier_ID_in NUMBER,
    Order_date_in DATE,
    Quantity_in NUMBER,
    Status_in VARCHAR2
)
IS
    v_order_id NUMBER;
BEGIN
    SELECT order_seq.NEXTVAL INTO v_order_id FROM dual;

    INSERT INTO Orders (Order_ID, Supplier_ID, Order_date,
        Quantity, Status)
    VALUES (v_order_id, Supplier_ID_in, Order_date_in, Quantity_in,
        Status_in);
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Ошибка при вставке данных: ' || SQLERRM);
    ROLLBACK;
END AddOrder;

CREATE OR REPLACE PROCEDURE DeleteOrder (
    v_order_id IN NUMBER
)
IS
    v_deleted_count NUMBER;
BEGIN
    DELETE FROM Orders WHERE Order_ID = v_order_id RETURNING
    COUNT(*) INTO v_deleted_count;

    IF v_deleted_count = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'No order found with this ID');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Order was deleted successfully');
    COMMIT;
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLCODE || SQLERRM);
    END;

CREATE OR REPLACE PROCEDURE UpdateOrder (
    u_order_id IN NUMBER,
    new_supplier_id IN NUMBER DEFAULT NULL,
    new_order_date IN DATE DEFAULT NULL,

```

```

new_quantity IN NUMBER DEFAULT NULL,
new_status IN VARCHAR2 DEFAULT NULL
)
IS
BEGIN
DECLARE
order_count NUMBER;
BEGIN
SELECT COUNT(*) INTO order_count
FROM Orders
WHERE Order_ID = u_order_id;

IF order_count = 0 THEN
raise_application_error(-20001, 'Order does not exist');
END IF;

IF new_supplier_id IS NOT NULL THEN
UPDATE Orders
SET Supplier_ID = new_supplier_id
WHERE Order_ID = u_order_id;
DBMS_OUTPUT.PUT_LINE('Supplier was updated. New supplier is: ' ||
new_supplier_id);
END IF;

IF new_order_date IS NOT NULL THEN
UPDATE Orders
SET Order_date = new_order_date
WHERE Order_ID = u_order_id;
DBMS_OUTPUT.PUT_LINE('Date was updated. New date is: ' ||
new_order_date);
END IF;

IF new_quantity IS NOT NULL THEN
UPDATE Orders
SET Quantity = new_quantity
WHERE Order_ID = u_order_id;
DBMS_OUTPUT.PUT_LINE('Quantity was updated. New quantity is: ' ||
new_quantity);
END IF;

IF new_status IS NOT NULL THEN
UPDATE Orders
SET Status = new_status
WHERE Order_ID = u_order_id;
DBMS_OUTPUT.PUT_LINE('Status was updated. New status is: ' ||
new_status);
END IF;

COMMIT;
DBMS_OUTPUT.PUT_LINE('Order updated successfully');
EXCEPTION

```

```

WHEN OTHERS THEN
ROLLBACK;
DBMS_OUTPUT.PUT_LINE('Error: ' || SQLCODE || SQLERRM);
END;
END;

CREATE OR REPLACE PROCEDURE AddProduct (
    p_user_login IN VARCHAR2,
    p_user_password IN VARCHAR2,
    p_beer_name IN VARCHAR2,
    p_product_quantity IN NUMBER,
    p_order_date IN DATE
)
IS
    v_user_id NUMBER;
    v_beer_id NUMBER;
    v_order_id NUMBER := user_orders_seq.NEXTVAL;
BEGIN
    IF AuthenticateUser(p_user_login, p_user_password) THEN
        SELECT user_id INTO v_user_id FROM User_info WHERE user_login =
p_user_login;

        SELECT beer_id INTO v_beer_id
        FROM (
            SELECT beer_id
            FROM Beer
            WHERE Beer_Name = p_beer_name
            ORDER BY DBMS_RANDOM.VALUE
        )
        WHERE ROWNUM = 1;

        INSERT INTO User_orders (users_order_id, user_id, ordered_product,
product_quantity, order_status, user_order_date)
VALUES (v_order_id, v_user_id, v_beer_id, p_product_quantity,
'active', p_order_date);

        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Order added successfully!' || 'Your order id
is ' || v_order_id);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Authentication failed.');
```

```

    END IF;
    EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('User or product not found.');
```

```

    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred.');
```

```

    END;

    CREATE OR REPLACE PROCEDURE DeleteProduct (
        p_user_login IN VARCHAR2,
```

```

        p_user_password IN VARCHAR2,
        p_order_id IN NUMBER
    )
    IS
        v_user_id NUMBER;
        v_order_status VARCHAR2(20);
    BEGIN
        IF AuthenticateUser(p_user_login, p_user_password) THEN
            SELECT user_id INTO v_user_id FROM User_info WHERE user_login =
            p_user_login;

            SELECT order_status INTO v_order_status
            FROM User_orders
            WHERE users_order_id = p_order_id
            AND (user_id = v_user_id) -- 1 is the admin user_id
            FOR UPDATE; -- Lock the row for update

            IF v_order_status = 'active' THEN
                DELETE FROM User_orders WHERE users_order_id = p_order_id;
                COMMIT;
                DBMS_OUTPUT.PUT_LINE('Order deleted successfully!');
            ELSE
                DBMS_OUTPUT.PUT_LINE('Order cannot be deleted. Status is not
                active.');
```

```

            END IF;
        ELSE
            DBMS_OUTPUT.PUT_LINE('Authentication failed.');
```

```

        END IF;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Order or user not found.');
```

```

        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('An error occurred.');
```

```

    END;

CREATE OR REPLACE PROCEDURE UpdateProduct (
    p_user_login IN VARCHAR2,
    p_user_password IN VARCHAR2,
    p_order_id IN NUMBER,
    p_product_quantity IN NUMBER,
    p_ordered_product_name IN VARCHAR2
)
    IS
        v_order_owner_id NUMBER;
        v_user_id NUMBER;
        v_ordered_product_id NUMBER;
    BEGIN
        IF AuthenticateUser(p_user_login, p_user_password) THEN
            SELECT user_id INTO v_user_id FROM User_info WHERE user_login =
            p_user_login;
```

```

SELECT user_id INTO v_order_owner_id
FROM User_orders
WHERE users_order_id = p_order_id
AND user_id = v_user_id;

IF v_order_owner_id IS NOT NULL THEN
SELECT Beer_ID INTO v_ordered_product_id
FROM Beer
WHERE Beer_Name = p_ordered_product_name
AND ROWNUM = 1;

UPDATE User_orders
SET product_quantity = p_product_quantity,
ordered_product = v_ordered_product_id
WHERE users_order_id = p_order_id
AND order_status = 'active';

IF SQL%ROWCOUNT = 0 THEN
DBMS_OUTPUT.PUT_LINE('No rows updated. The order might be inactive
or not owned by the user. ');
ELSE
DBMS_OUTPUT.PUT_LINE('Order updated successfully!');
END IF;
ELSE
DBMS_OUTPUT.PUT_LINE('The order does not exist or is not owned by
the user. ');
END IF;
ELSE
DBMS_OUTPUT.PUT_LINE('Authentication failed. ');
END IF;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('No order found for the user. ');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;

CREATE OR REPLACE PROCEDURE ProductAnalysis (
    start_date IN DATE,
    end_date IN DATE
)
IS
    v_total_quantity NUMBER;
BEGIN
    IF end_date < start_date THEN
        DBMS_OUTPUT.PUT_LINE('Ошибка: Некорректные даты');
        RETURN;
    END IF;

    SELECT SUM(product_quantity) INTO v_total_quantity
    FROM User_orders

```

```

WHERE user_order_date BETWEEN start_date AND end_date;

DBMS_OUTPUT.PUT_LINE('Общее количество продукции за период с ' ||
start_date || ' по ' || end_date || ': ' || v_total_quantity || '
штук');

DBMS_OUTPUT.PUT_LINE('Анализ продукции за период с ' || start_date
|| ' по ' || end_date || ':');
FOR product IN (
SELECT ordered_product, SUM(product_quantity) AS total_quantity
FROM User_orders
WHERE user_order_date BETWEEN start_date AND end_date
GROUP BY ordered_product
)
LOOP
DBMS_OUTPUT.PUT_LINE('Продукт ' || product.ordered_product || ': '
|| product.total_quantity || ' штук');
END LOOP;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Данные за указанный период отсутствуют');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Произошла ошибка: ' || SQLERRM);
END;

CREATE OR REPLACE PROCEDURE RegisterUser(
    p_user_name IN VARCHAR2,
    p_user_password IN VARCHAR2,
    p_repeat_password IN VARCHAR2,
    p_birthday IN DATE,
    p_user_login IN VARCHAR2,
    p_result OUT VARCHAR2
)
IS
login_exists NUMBER;
hashed_password RAW(256); --SHA-256
BEGIN
IF p_user_password != p_repeat_password THEN
p_result := 'Passwords do not match.';
ELSE
SELECT COUNT(*) INTO login_exists FROM User_info WHERE user_login =
p_user_login;

IF login_exists > 0 THEN
p_result := 'User login already exists.';
ELSE
hashed_password :=
DBMS_CRYPTO.HASH(UTL_RAW.CAST_TO_RAW(p_user_password),
DBMS_CRYPTO.HASH_SH256);

```

```
INSERT INTO User_info (user_id, user_name, user_password, birthday,
user_login)
VALUES (user_info_seq.NEXTVAL, p_user_name, hashed_password,
TO_DATE(p_birthday, 'DD-MM-YYYY'), p_user_login);

IF SQL%ROWCOUNT > 0 THEN
p_result := 'User registered successfully!';
ELSE
p_result := 'User registration failed.';
END IF;
END IF;
END IF;
EXCEPTION
WHEN OTHERS THEN
p_result := 'An error occurred: ' || SQLERRM;
END;
```