МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Информационные системы и технологии
Специальность 1–40 05 01 Информационные системы и технологии
Специализация 1-40 05 01-03 Информационные системы и технологии (издательско-полиграфический комплекс)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

«Разработка базы данных “Банк крови” с применением технологии резервного копирования и восстановления данных»

Выполнил студент Позняк Полина Павловна
(Ф.И.О.)

Руководитель проекта асс., Сазонова Д.В.
(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В .
(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: асс., Сазонова Д.В.
(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: асс., Сазонова Д.В.
(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой _____

Минск 2023

Оглавление

Введение

Банк крови – это неотъемлемая часть системы здравоохранения, обеспечивающая доступность крови для лечения пациентов с различными заболеваниями и травмами. Однако, эффективное управление банком крови требует надежной базы данных, которая позволяет отслеживать все процессы – от поступления крови до ее выдачи пациентам.

Целью данной работы являлась разработка реляционной базы данных, представляющей собой систему управления банка крови. Эта база данных составлялась для обеспечения возможности управления транзакциями крови между донорами и реципиентами, поиск подходящих доноров в случай необходимости, а также отслеживания доступных ресурсов крови в банках крови.

В качестве СУБД для базы данных была использована Oracle 12c. Выбранной технологией является резервное копирование и восстановление данных в связи с важностью хранимых медицинских данных.

Основными требованиями к работе являлись:

– реализация ролей администратора и пользователя;

– взаимодействие с базой данных при помощи хранимых процедур;

– реализация выбранной технологии;

– тестирование производительности базы данных;

– импорт данных из XML файлов, экспорт данных в формат XML.

В пояснительной записке вы сможете найти краткую информацию о похожих проектах, о реализации базы данных и руководство пользователя к разработанному приложению.

1 Аналитический обзор литературы

1.1 Изучение требований, определение вариантов использования

Требования для курсовой работы по базе данных "Банк крови":

1. Описание структуры базы данных: должна быть описана структура таблиц, их связи и поля.
2. Обоснование выбора СУБД: необходимо обосновать выбор конкретной системы управления базами данных, объяснить, почему именно она была выбрана для решения задачи.
3. Задание целей проекта: нужно четко определить цели, которые должны быть достигнуты при создании базы данных "Банк крови".
4. Описание функционала базы данных: необходимо описать функции, которые должна выполнять база данных, чтобы эффективно решать поставленные цели.
5. Описание сценариев использования базы данных: нужно описать сценарии использования базы данных, т.е. ситуации, в которых она может быть применена.

Варианты использования базы данных "Банк крови":

1. Регистрация доноров: база данных может быть использована для регистрации доноров и хранения информации о них, такой как имя, группа крови с резус-фактором, адрес проживания и т.д.
2. Хранение информации о крови: база данных может содержать информацию о всех пожертвованных единицах крови, включая группу крови с резус-фактором, имеющийся объем и т.д.
3. Поиск доноров: база данных может быть использована для поиска подходящих доноров по группе крови с резус-фактором.
4. Контроль состояний доноров: если донор сдавал кровь в недавнем времени, необходимо помечать его, как временно недоступного для повторной сдачи крови.
5. Управление запасами крови: база данных может использоваться для управления запасами крови и контроля их доступности в банке крови.
6. Статистический анализ: база данных может быть использована для сбора и анализа статистических данных о донорах, пожертвованной крови и использовании запасов крови.

1.2 Анализ аналогичных решений

"Банк крови" – это специализированное приложение, предназначенное для организации хранения, поиска и выдачи донорской крови. Аналоги такого приложения могут быть представлены различными медицинскими и социальными платформами, которые предлагают услуги поиска и координации доноров крови. Некоторые из них рассмотрены ниже:

National Blood Transfusion Service. Это сетевое (LAN) программное обеспечение представляет собой ИТ-решение для банка крови для поддержки повседневных действий. Программное обеспечение написано с использованием языка Java, а пользовательский интерфейс реализован с помощью JavaFX [1].

Достоинства:

1. Использование паттерна Observer Design – кровь запрашивает балансировку нагрузки и связь между палатами и лабораториями.

2. Шаблон проектирования резервирования – задачи управления в условиях сильного параллелизма.

2. Простота интерфейса.

Недостатки:

1. Данное программное обеспечение разработано в основном для образовательных целей.

2. Приложение только под Linux.

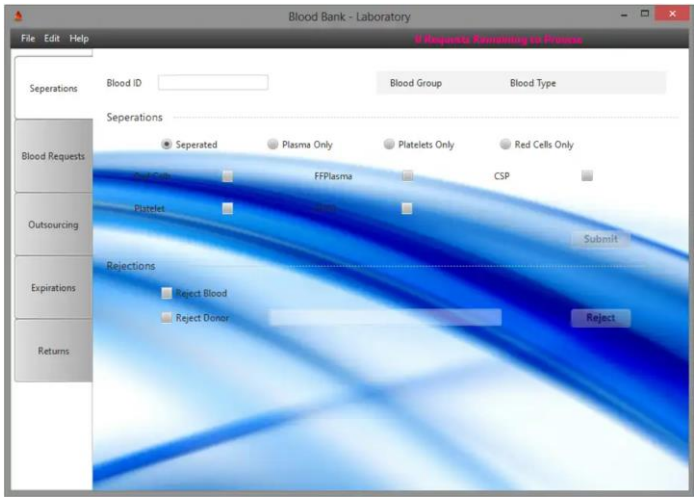

Рисунок 1 – Скриншот интерфейса приложения «Blood Bank»

Blood4Life. Приложение Blood4Life в основном используется для людей, которым нужна кровь в любых чрезвычайных ситуациях, а также для тех, кто хочет помочь другим, сдав кровь. Пользователи могут осуществлять поиск по списку доноров на основе области. В случае чрезвычайной ситуации предоставляется возможность быстрого поиска. В режиме быстрого поиска человек, нуждающийся в крови, может найти донора крови без предварительного входа в систему. После получения результатов о доступном доноре крови для любой конкретной области, человек, нуждающийся в крови может просмотреть профиль и связаться с ним по телефону, СМС или электронной почте [2].

Достоинства:

1. Режим быстрого доступа. Можно осуществлять поиск крови в приложении как авторизированным, так и не авторизированным пользователям.

2. Поиск по конкретному району. Лица, ищущие кровь, могут найти донора крови в ближайшем к ним районе.

3. Простота интерфейса.

Недостатки:

1. Локальность. Поиск только крови только по ограниченному числу районов.

2. Устаревший интерфейс.

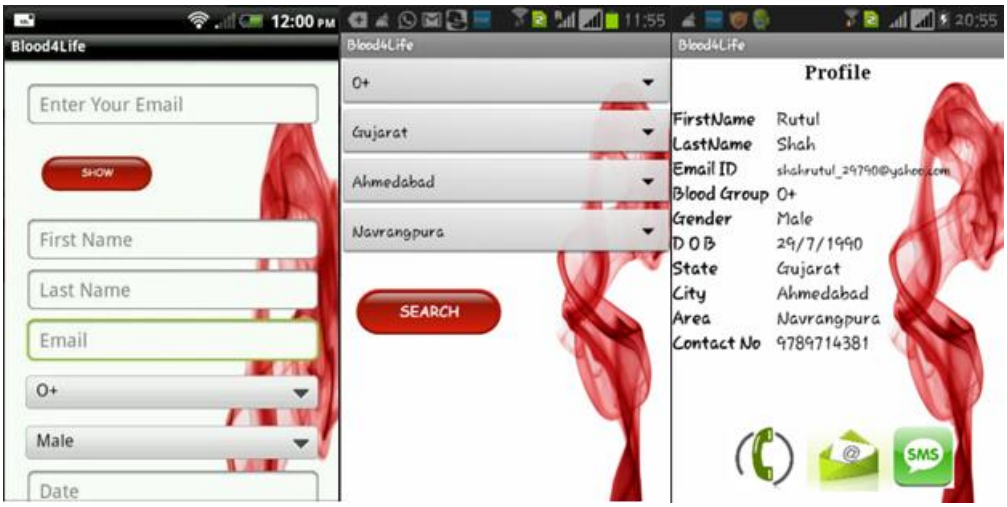

Рисунок 2 – Скриншот интерфейса приложения «Blood4Life»

Рассмотренные приложения и платформы предоставляют аналогичные услуги "Банка крови", и каждое из них может быть использовано для поиска и координации доноров крови. Однако, конкретный выбор приложения может зависеть от места проживания, доступности центров сбора крови и других факторов.

1.3 Вывод по разделу

После изучения требований, обзора вариантов использования и анализа аналогичных решений были сформированы соответствующие требования к разрабатываемому курсовому проекту. Было решено, что приложение будет представлять собой инструмент для медицинского учреждения, позволяющий администратору (главному врачу) получать полное управление данными о донорах, получателях крови, банках крови. А обычный пользователь (медицинский работник) будет иметь право на просмотр данных о донорах, получателях крови и банках крови, а также на ввод некоторых новых записей в базу данных и просмотра некоторой статистики.

Таким образом, для нашего приложения, которое будет рассчитано на использование медицинскими работниками, мы можем предположить, какие объекты необходимо создать для модели нашей БД, разработка которой описана в следующем разделе.

2 Проектирование базы данных

2.1 Разработка модели базы данных

После анализа информации, которую необходимо хранить в нашей базе данных, мы организуем их в таблицы и создадим логические связи между ними с помощью первичных (PK) и внешних ключей (FK).

Диаграмма базы данных, спроектированной изображена на рисунке 3.

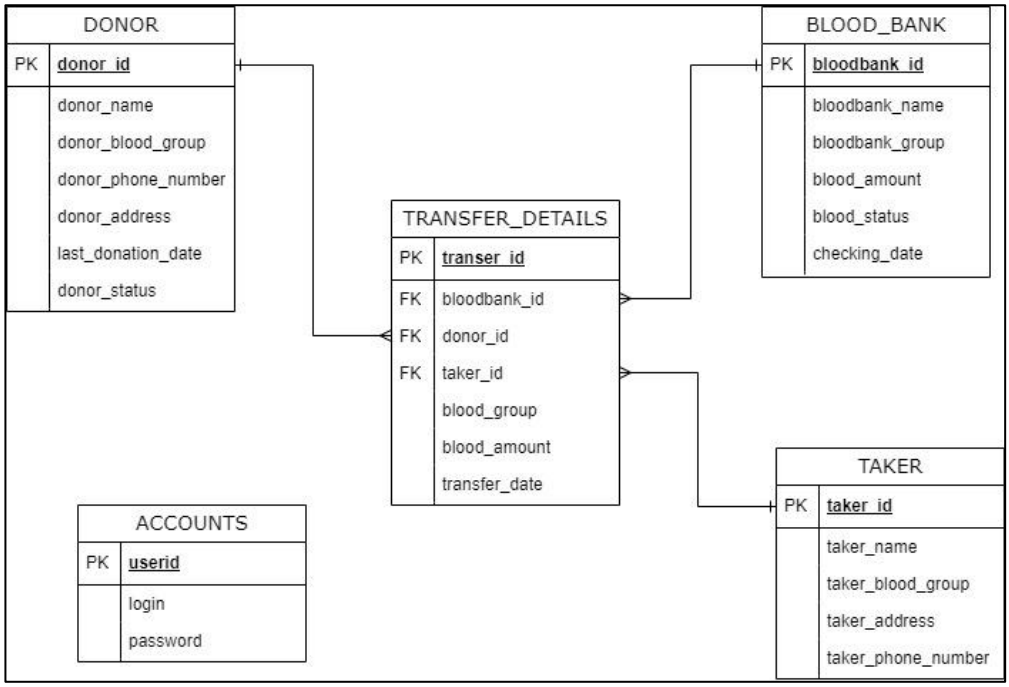

Рисунок 3 – Диаграмма базы данных

База данных приложения состоит из 5 таблиц, взаимосвязанных между собой внешними ключами, соответствующими полями-ID каждой из таблиц:

– Donor;
– Taker;
– Transfer_Details;
– Blood_Bank;
– Account.

Скрипт для создания базы данных приведен в приложении А. Однако, прежде чем выполнять данный скрипт, необходимо так же разработать инфраструктуру и возможные триггеры на таблицы, создание которых рассмотрится далее.

2.2 Описание информационных объектов и ограничений целостности

2.2.1 Таблица донора

Для реализации базы данных «Банк крови» было разработано 5 таблиц: «Donor», «Taker», «Transfer_Details», «Blood_Bank», «Account». Каждая из которых рассмотрена ниже более подробно.

Таблица «Donor» содержит контактную информацию о донорах, которые могут сдать кровь.

Таблица 1 – Таблица «Donor»

Наименование	Тип	Ограничение
donor_id	number	PK
donor_name	varchar(50)	NOT NULL
donor_blood_group	varchar(20)	NOT NULL
donor_phone_number	varchar(20)	
donor_address	varchar(50)	NOT NULL
last_donation_date	date	NOT NULL
donor_status	varchar(20)	

Подробное описание каждого из столбцов таблицы «Donor»:

– donor_id – уникальный идентификатор донора, генерируемый автоматически с помощью ключевого слова GENERATED ALWAYS AS IDENTITY и являющийся первичным ключом таблицы;

– donor_name – имя донора, заданное в виде строки с максимальной длиной 50 символов;

– donor_blood_group – группа крови донора, заданная в виде строки с максимальной длиной 20 символов;

– donor_phone_number – номер телефона донора, заданный в виде строки с максимальной длиной 20 символов, являющийся не обязательным, то есть может быть пустым;

– donor_address – адрес донора, заданный в виде строки с максимальной длиной 50 символов;

– last_donation_date – дата последней сдачи крови донором;

– donor_status – статус донора, заданный в виде строки с максимальной длиной 20 символов, являющийся не обязательным, то есть может быть пустым.

2.2.2 Таблица получателя крови

Таблица «Taker» содержит информацию о получателях крови. Все типы данных для соответствующих столбцов и ограничения целостности приведены в таблице 2.

Таблица 2 – Таблица «Taker»

Наименование	Тип	Ограничение
taker_id	number	PK
taker_name	varchar(50)	NOT NULL
taker_blood_group	varchar(20)	NOT NULL
taker_address	varchar(50)	NOT NULL
taker_phone_number	varchar(20)	

Подробное описание каждого из столбцов таблицы «Taker»:

– taker_id – уникальный идентификатор получателя, генерируемый автоматически с помощью ключевого слова GENERATED ALWAYS AS IDENTITY. и являющийся первичным ключом таблицы;

– taker_name – имя получателя, заданное в виде строки с максимальной длиной 50 символов;

– taker_blood_group – кровная группа получателя, заданная в виде строки с максимальной длиной 20 символов;

– taker_address – адрес получателя, заданный в виде строки с максимальной длиной 50 символов;

– taker_phone_number – номер телефона получателя, заданный в виде строки с максимальной длиной 20 символов, являющийся не обязательным, то есть может быть пустым.

2.2.4 Таблица банка крови

Таблица «Blood_Bank» содержит информацию о запасах крови в банке крови.

Все типы данных для соответствующих столбцов и ограничения целостности приведены в таблице 3.

Таблица 3 – Таблица «Blood_Bank»

Наименование	Тип	Ограничение
bloodbank_id	varchar(10)	PK
bloodbank_name	varchar(50)	NOT NULL
blood_group	varchar(20)	NOT NULL
blood_amount	float	
blood_status	varchar(20)	
checking_date	date	

Подробное описание каждого из столбцов таблицы «Blood_Bank»:

– bloodbank_id – уникальный идентификатор банка крови, заданный в виде строки с максимальной длиной 10 символов и являющийся первичным ключом таблицы;

– bloodbank_name – название банка крови, заданное в виде строки с максимальной длиной 50 символов;

– blood_group – кровная группа крови, заданная в виде строки с максимальной длиной 20 символов. Это является внешним ключом, связывающим таблицу Blood_Bank с таблицей Transfer_Details;

– blood_amount – количество крови в миллилитрах, доступное в банке крови. Задано в виде числа с плавающей точкой;

– blood_status – статус крови в банке крови, заданный в виде строки с максимальной длиной 20 символов. Например, кровь может быть свежей, храниться дольше времени или не подходить для использования;

– checking_date – дата последней проверки крови, заданная в виде типа данных DATE.

2.2.5 Таблица транзакции крови

Таблица «Transfer_Details» содержит информацию о передаче крови от донора к получателю через банк крови.

Таблица 4 – Таблица «Transfer_Details»

Наименование	Тип	Ограничение
transfer_id	number	PK
bloodbank_id	varchar(10)	FK
donor_id	number	FK
taker_id	number	FK
blood_group	varchar(50)	NOT NULL
blood_ampunt	float	NOT NULL
transfer_date	date	

Подробное описание каждого из столбцов таблицы «Transfer_Details»:

– transfer_id – уникальный идентификатор передачи крови, генерируемый автоматически, заданный в виде числа;

– bloodbank_id – уникальный идентификатор банка крови, связанный с передачей крови. Это внешний ключ, связывающий таблицу Transfer_Details с таблицей Blood_Bank;

– donor_id – уникальный идентификатор донора, связанный с передачей крови. Это внешний ключ, связывающий таблицу Transfer_Details с таблицей Donor.

– taker_id - уникальный идентификатор получателя, связанный с передачей крови. Это внешний ключ, связывающий таблицу Transfer_Details с таблицей Taker;

– blood_group - кровная группа крови, передаваемая от донора к получателю, заданная в виде строки с максимальной длиной 20 символов;

– blood_amount - количество крови в миллилитрах, передаваемое от донора к получателю. Задано в виде числа с плавающей точкой;

– transfer_date - дата передачи крови, заданная в виде типа данных DATE.

2.2.6 Таблица аккаунта пользователя

Таблица «Account» содержит информацию о пользователях системы. Она имеет следующие столбцы:

Таблица 5 – Таблица «Account»

Наименование	Тип	Ограничение
userid	number	PK
login	varchar2(30)	NOT NULL
password	nvarchar2(100)	

Подробное описание каждого из столбцов таблицы «Account»:

– userid – числовой столбец, который является первичным ключом таблицы. Значения этого столбца генерируются автоматически идентификатором;

– login – строковый столбец, который содержит логины пользователей. Значения в этом столбце должны быть уникальными и не могут быть пустыми (not null);

– password – строковый столбец, который содержит пароли пользователей. Значения в этом столбце могут быть пустыми (null) и хранятся в формате NVARCHAR2 с длиной 100 символов.

2.3 Разработка объектов базы данных

2.3.1 Пользователи

Для определения числа юзеров, необходимо понимать, что подразумевает собой термин «пользователь». Пользователь базы данных – это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации.

В зависимости от назначения приложения может быть различное число пользователей. Так как разрабатываемая база данных рассчитана на использование в рамках медицинского учреждения, то администратором может быть главный врач, который имеет полное право на управление данными о донорах, получателях крови, банках крови. И обычный пользователь-врач, с ограниченным набором прав, предоставляющий соответствующие возможности лишь на просмотр.

В базе разрабатываемой базе данных формируются два пользователя базы данных (главный врач-админ и пользователь-врач) и множество пользователей приложения (аккаунты), которые будут являться фактическими пользователями-врачами с ограниченными правами на выполнение процедур.

Скрипт для создания инфраструктуры базы данных, пользователей и назначения им ролей и привилегий представлен в приложении Б.

2.3.2 Триггеры базы данных

Для базы данных созданы три триггера: TR_DONOR, TR_BANK, TR_BANK2.

Триггер TR_DONOR создан для обновления статуса донора в таблице «Donor». Триггер срабатывает при вставке или обновлении строк в таблице «Donor» для каждой строки (FOR EACH ROW).

В теле триггера производится проверка на количество дней, прошедших с даты последней донорской крови (:new.last_donation_date) до текущей даты (systimestamp). Если это количество дней меньше 90, то статус донора (:new.donor_status) устанавливается как «Not Available», что означает, что этот донор не может быть использован для переливания крови. В противном случае статус донора устанавливается как «Available», что означает, что этот донор может быть использован для переливания крови.

Таким образом, данный триггер автоматически обновляет статус донора в таблице «Donor» на основе даты последней донорской крови и текущей даты.

Триггер TR_BANK создан для автоматического обновления статуса крови в банке на основе количества единиц крови. Триггер запускается перед выполнением операций INSERT или UPDATE на таблице «Blood_Bank» и выполняется для каждой строки, которая будет добавлена или изменена.

Триггер начинается с проверки количества крови, которое будет добавлено или изменено в таблице. Если количество крови меньше 1, то статус крови будет изменен на «Not Available». Если количество крови больше 1, но меньше 20, то статус будет изменен на «Only For Emergency». Если количество крови больше 20, но меньше 100, то статус будет изменен на «Good Collection». Если количество крови больше 100, то статус будет изменен на «Adequate». В результате, статус

крови в банке будет автоматически обновляться в соответствии с количеством крови, что поможет медицинским работникам легче отслеживать доступность крови в банке.

Триггер TR_BANK2 создан для проверки корректности вставляемых значений в столбец blood_group таблицы «Blood_Bank».

Триггер запускается перед операцией вставки или обновления в таблице Blood_Bank для каждой строки. Если значение столбца blood_group является одним из допустимых значений (A+, A-, B+, B-, AB+, AB-, O+, O-), тогда выводится сообщение Blood group insertion is Okay. В противном случае, вызывается ошибка «Incorrect Blood Group Insertion» с кодом -20000, которая прерывает операцию вставки или обновления.

Скрипт создания триггеров приводится в приложении В.

2.3.3 Процедуры базы данных

Для управления данными через приложение пользователи и администратор использует хранимые процедуры. Хранимая процедура – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере.

Для базы данных курсового проекта были разработаны несколько типов процедур.

1) Процедуры выборки данных из таблиц без параметров (разработанные процедуры представлены в приложении Г):

– GETDONORS;
– GETTAKERS;
– GETBLOOD_BANKS
– GETTRANSFER_DETAILS;
– GETTRANSFER_DETAILS_FOR_CLIEN;
– GETACCOUNTS.

2) Процедуры выборки данных из таблиц с параметрами (приложение Д):

– FIND_DONORS_BY_BLOODGROUP;
– FIND_AVAIL_DONORS_BY_GR_ADDR;
– GET_BLOOD_LEADER;
– FIND_UNAVAIL_DONORS_BY_GR_ADDR.

3) Процедуры добавление данных в таблицы (основной задачей которых является внесение новых записей данных каждой сущности в соответствующие таблицы):

– ADD_NEW_DONOR;
– ADD_NEW_TAKER;
– ADD_NEW_BLOOD_BANK;
– ADD_NEW_TRANSFER_DETAILS;
– ADD_NEW_ACCOUNT.

4) Процедуры изменения данных в таблицах (основной задачей которых является внесение изменений в существующих записях таблиц базы данных):

– UPDATE_DONOR_INFO;
– UPDATE_TAKER_INFO;

– UPDATE_BLOOD_BANK;
– UPDATE_TRANSFER_DETAILS;
– UPDATE_ACCOUNT.

5) Процедуры удаления данных из таблиц:
– DELETE_DONOR;
– DELETE_TAKER;
– DELETE_BLOOD_BANK;
– DELETE_TRANSFER_DETAILS;
– DELETE_TRANSF_DET_BY_PARAMS;
– DELETE_ACCOUNT.

Данные процедуры позволяют выполнять основные операции добавления, обновления, удаления и получения данных (приложение Е), а также получение нескольких статистических сведений, например, определение лидера среди доноров в течение промежутка времени, заданного пользователем.

2.3.4 Функции базы данных

В Oracle, функции – это объекты базы данных, которые могут принимать ноль или более аргументов в качестве входных параметров и возвращать единственное значение. Функции могут использоваться в выражениях SQL, PL/SQL, возвращая результат, который может быть дальше использован в различных операциях.

Для базы данных курсовой работы было разработано три функции для получения статистических данных:

1) Функция функция TOTAL_BLOOD_TRANSFER принимает в качестве параметра группу крови (p_blood_group) и возвращает суммарное количество переливаемой крови для этой группы. Функция выполняет запрос на выборку суммарного объема крови из таблицы Blood_Bank, где группа крови равна переданному параметру, и возвращает эту сумму.

2) Функция NOTAL_NO принимает в качестве параметра дату (transfer_date_in) и возвращает количество переливаний крови, произведенных в этот день. Функция выполняет запрос на выборку количества записей в таблице Transfer_Details, где дата переливания равна переданному параметру, и возвращает это количество.

3) Функция BLOOD_TRANSFERS_VOLUME_ON_DATE принимает в качестве параметра дату (transfer_date) и возвращает объем переливаемой крови в этот день. Функция выполняет запрос на выборку суммарного объема крови из таблицы Transfer_Details, где дата переливания равна переданному параметру, и возвращает этот объем.

Разработанные функции, предоставляющие статистические данные, представлены в приложении Ж.

3 Описание процедур импорта и экспорта

В курсовом проекте разработаны процедуры импорта и экспорта данных для таблицы «Taker» (приложение 3).

Процедура EXPORT_TAKERS_TO_XML создает XML-файл, содержащий данные из таблицы «Taker». Сначала создаются директории "EXPORT_DATA" и "IMPORT_DATA", указывающие путь к папкам, в которые будут экспортироваться и импортироваться данные соответственно. Затем создается курсор «xmlcur», который выбирает данные из таблицы «Taker» и создает XML-элементы для каждой записи. В конце цикла собираются все элементы в один элемент «Taker», который содержит все записи из таблицы «Taker». После этого процедура создает XML-файл и записывает в него данные из курсора «xmlcur» с помощью функции «DBMS_XMLDOM.WRITETOFILE». Файл будет сохранен в директории «EXPORT_DATA» с именем «Takers.xml». Таким образом, процедура экспортирует данные из таблицы «Taker» в XML-файл, который можно использовать для обмена данными между различными системами или для сохранения данных в формате, который может быть легко использован с другими приложениями.

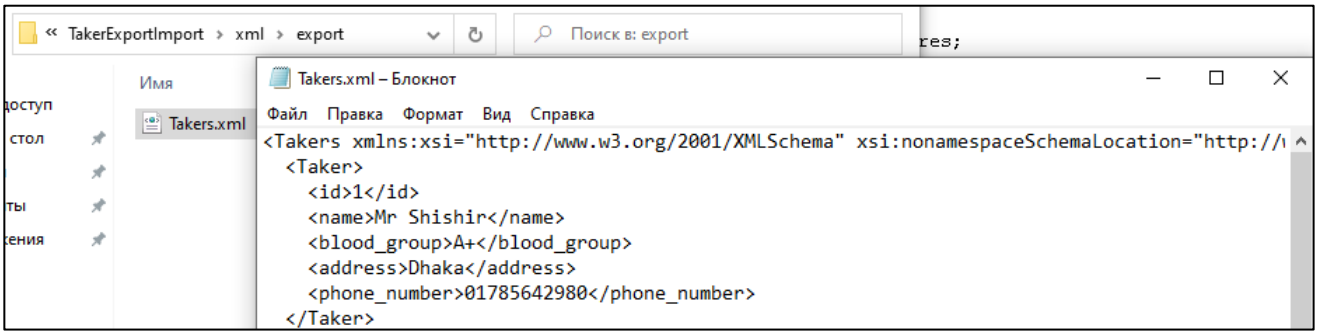

Рисунок 4 – Результат выполнения процедуры EXPORT_TAKERS_TO_XML

Процедура IMPORT_TAKERS_FROM_XML импортирует данные из XML-файла в таблицу «Taker» базы данных Oracle. В начале процедуры создаются необходимые переменные и открывается файл XML с данными. Далее создается XML-парсер, который парсит содержимое XML-файла. Полученные данные добавляются в соответствующие поля структуры «tk» типа Taker%ROWTYPE. Затем используется оператор INSERT, чтобы добавить полученные данные в таблицу «Taker». После завершения цикла обработки всех записей в XML-файле, закрываются временные CLOB- и XML-объекты. После выполнения процедуры данные будут импортированы из XML-файла в таблицу «Taker».

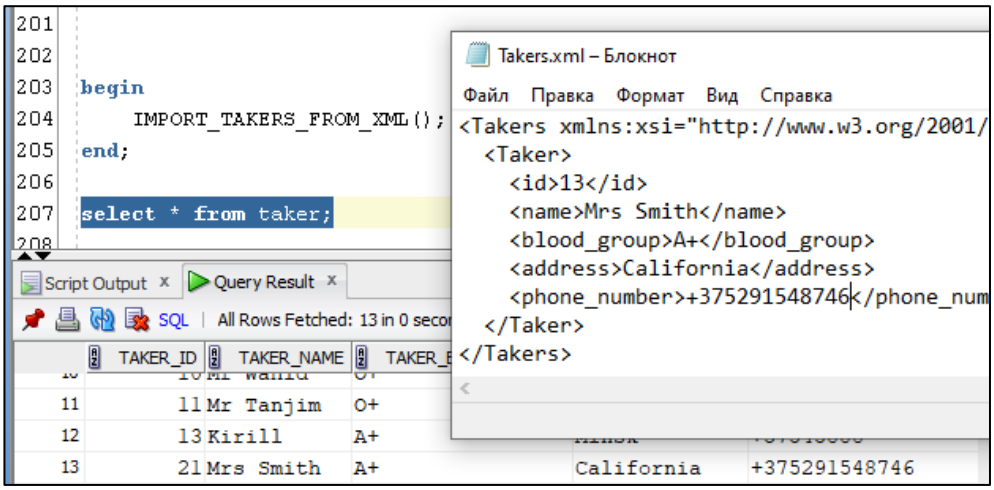

Рисунок 5 – Результат выполнения процедуры IMPORT_TAKERS_TO_XML

Данные процедуры импорта и экспорта в формате XML в базе данных Oracle, могут быть полезны в нескольких сценариях.

Во-первых, импорт и экспорт в XML могут использоваться для обмена данными между различными системами или приложениями, которые могут использовать XML как стандартный формат обмена данными.

Во-вторых, функции импорта и экспорта в XML могут быть полезны при миграции данных из одной базы данных Oracle в другую.

В-третьих, функции импорта и экспорта в XML могут использоваться для сохранения данных в формате, который может быть легко использован с другими приложениями, которые могут не быть совместимы с Oracle базой данных.

4 Тестирование производительности

Производительность базы данных означает ее способность быстро и эффективно обрабатывать запросы и транзакции.

Высокая производительность базы данных может быть достигнута с помощью эффективного использования индексов, хорошей организации структуры базы данных и оптимизации запросов.

Чтобы проверить производительность базы данных, нужно заполнить ее разными данными в большом объеме и измерить время выполнения запроса. Для этой задачи был создан анонимный блок (листинг 1), который использует готовую процедуру для вставки данных в таблицу и позволяет добавить много строк за одно выполнение.

```
BEGIN
FOR Lcntr IN 1..100000
LOOP
   ADD_NEW_DONOR('Testing','A+','Minsk','12345','01-MAY-2023');
END LOOP;
END;
```

Листинг 1 – Анонимный блок для заполнения таблицы большим количеством данных

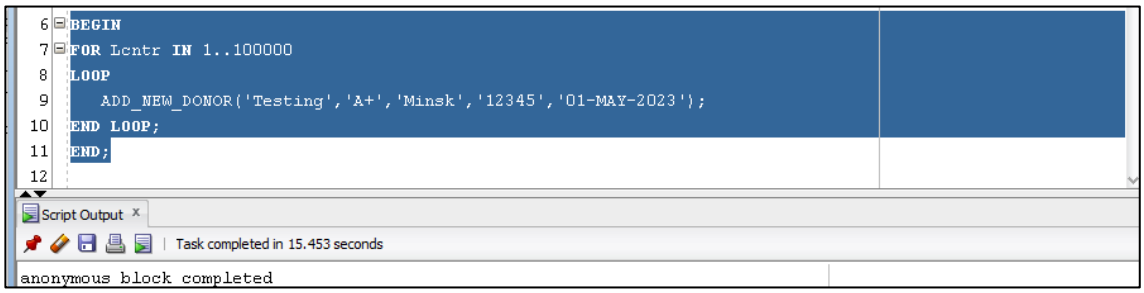

Рисунок 6 – Добавление 100000 строк в таблицу Donors.

Затем в другом анонимном блоке делается запрос, который должен вернуть 40000 строк, и измеряется время его выполнения с помощью переменной и стандартных средств Oracle для просмотра плана запроса (листинг 2).

```
declare
    start_time number := dbms_utility.get_time();
begin
    for r in (select * from Donor where donor_id > 80000 and donor_id
< 90000) loop null; end loop;
    dbms_output.put_line('Elapsed time: '||(dbms_utility.get_time() -
start_time)/100);
end;
```

Листинг 2 – Анонимный блок, измеряющий время получения данных

Результат выполнения анонимного блока представлен на рисунке 7. Также стоит отметить, что для просмотра плана запроса используются стандартные

средства Oracle, а именно кнопку на главной панели (либо клавишу F10), предварительно выделив данный запрос.

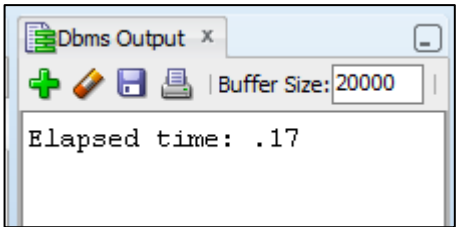

Рисунок 7 – Время выполнения анонимного блока.

После тестирования времени выполнения получения данных без оптимизации, теперь стоит найти решение, чтобы еще улучшить производительность, а именно создадим индекс и проанализировать время выполнения получения данных запроса с индексом.

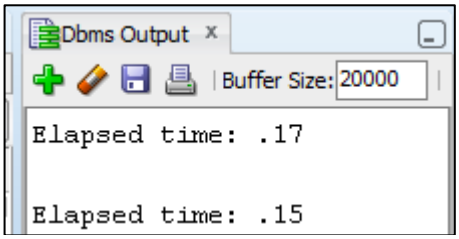

Рисунок 8 – Время выполнения анонимного блока после создания индекса.

Исходя из результата полученного на рисунке 1, время запроса уменьшилось, но тем не менее даже без индекса наша база данных успешно прошла тест на производительность.

Как видим, индексы позволяют ускорить поиск и выборку данных из базы данных, так как они позволяют эффективно находить нужные записи без полного сканирования таблицы.

В разрабатываемой базе данных целесообразно создание индексов на полях, которые часто используются для поиска и выборки данных (такие как donor_blood_group, taker_blood_group, donor_id, taker_id, blood_group), позволит ускорить операции выборки и фильтрации данных в таблицах «Donor», «Taker», «Blood_Bank» и «Transfer_Details».

Создание индекса на поле blood_group в таблице «Transfer_Details» также поможет ускорить операции выборки данных, связанных с группами крови, которые являются важным критерием для выборки при переливании крови.

Таким образом, создание индексов на этих полях позволит значительно повысить производительность запросов и ускорить обращение к данным в базе данных.

5 Описание технологии

Технология резервного копирования и восстановления данных является критически важной для базы данных Банка крови. База данных Банк крови содержит критически важную информацию о донорах, реципиентах, банках крови и транзакциях крови. В случае потери или повреждения данных, это может привести к непредсказуемым последствиям и серьезным проблемам.

Технология резервного копирования позволяет создавать копии данных, которые могут быть использованы для восстановления информации в случае потери или повреждения основных данных. Это позволяет сохранить доступность данных и обеспечить непрерывность бизнес-процессов.

В случае базы данных Банка крови, резервное копирование может помочь восстановить данные о донорах, получателях, банках крови и транзакциях крови в случае аварийного сбоя, ошибок пользователей или других нежелательных событий. Без технологии резервного копирования и восстановления данных, потеря данных может привести к серьезным проблемам, таким как нарушение прав доступа, утечка конфиденциальной информации и даже правовые последствия.

Поэтому, технология резервного копирования и восстановления данных является необходимым инструментом для обеспечения безопасности и непрерывности бизнес-процессов в базе данных Банка крови.

Для реализации технологии был использован Oracle Recovery Manager (RMAN) и выполнялась последовательность действий, описанная далее [3]. Листинг выполненных команд приведен в приложении И.

Прежде всего необходимо было создать директорию, где будут храниться резервные копии базы данных. Либо же еще её можно назвать область быстрого восстановления (FRA).

Далее запускаем утилиту SQLPlus в режиме суперпользователя, выполняем подключения к базе данных и запускаем базу данных (если она еще не запущена).

После чего запускаем утилиту RMAN и при помощи нее задаем переменные среды базы данных для области быстрого восстановления.

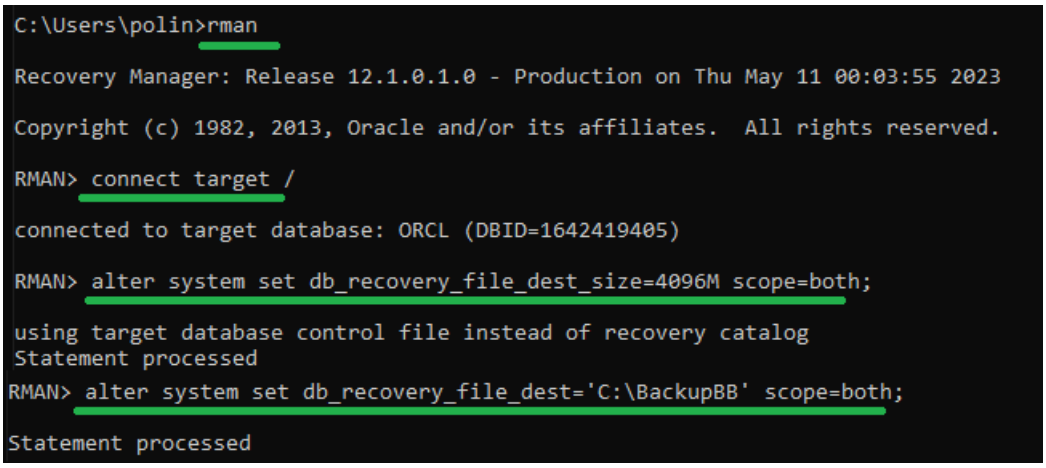

Рисунок 9 – Задание переменных среды базы данных для области быстрого восстановления при помощи утилиты RMAN.

Далее необходимо убедиться, что база данных находится в режиме журнала архивации, чтобы задействовать оперативное резервное копирование.

Если он находится в режиме NOARCHIVELOG, выполните команды SQLPlus, приведенные в листинге 3.

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE ARCHIVELOG;
SQL> ALTER DATABASE OPEN;
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

Листинг 3 – Процесс включения режима журналирования базы данных

После чего база данных резервируется с помощью команды «RMAN> backup as compressed backupset database plus archivelog;»

Затем удаляем DBF файлы и можем проверить, удостоверившись, что подключиться к базе данных не получится.

В для восстановления данных в CMD пишем следующее:

```
> rman
RMAN> connect target /
RMAN> shutdown abort
RMAN> startup mount
RMAN> restore database;
RMAN> recover database;
```

Листинг 4 – Процесс восстановления данных при помощи утилиты RMAN

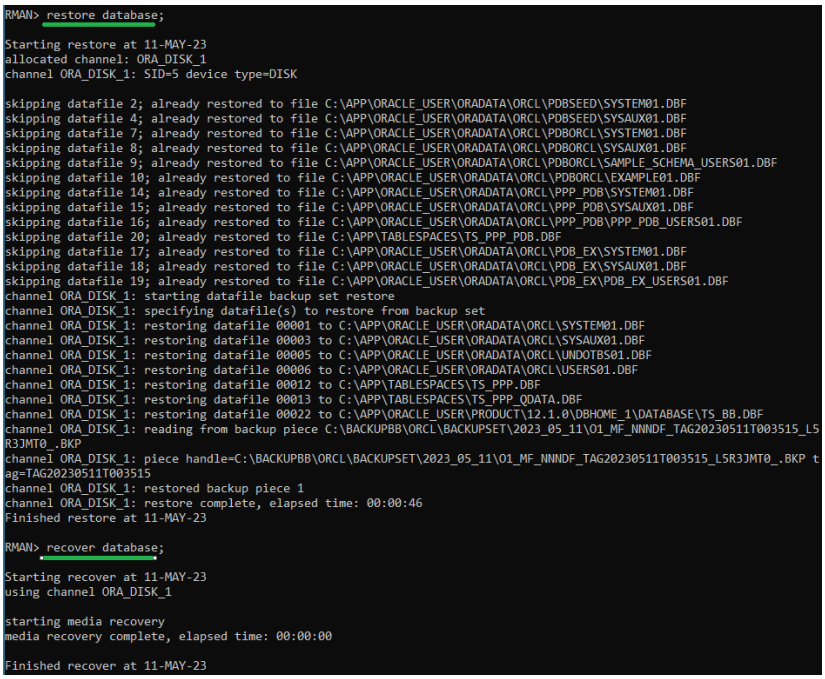

Рисунок 10 – Результат выполнения команд восстановления данных при помощи утилиты RMAN

Таким образом мы произвели создание резервной копии базы данных с помощью утилиты RMAN, удаление файлов DBF, что привело к полной неработоспособности базы данных. Однако, благодаря предварительно созданной резервной копии, при помощи команд RMAN для восстановления базы данных она была восстановлена до состояния, которое было до удаления файлов DBF. В конечном итоге, база данных успешно восстановлена и открыта для использования.

Использование утилиты RMAN в Oracle Database существенно упрощает управление резервным копированием и восстановлением данных, сокращает время простоя при восстановлении (позволяет восстанавливать только нужные объекты, а не всю базу данных), а также улучшает производительность и уменьшает вероятность ошибок.

6 Краткое описание приложения для демонстрации

Приложение для демонстрации работы спроектированной базы данных «Банк крови» было разработано на языке C# с применением технологии разработки пользовательских интерфейсов для приложений Windows WPF (Windows Presentation Foundation).

Для работы с СУБД Oracle был использован официальный .NET-драйвер представленный NuGet-пакетом Oracle.ManagedDataAccess. Данный драйвер позволяет создавать соединения с базой данных, выполнять запросы и получать результаты.

Oracle.ManagedDataAccess позволяет разработчикам использовать ORM-фреймворки, такие как Entity Framework, для работы с данными в базе данных Oracle. Этот пакет также содержит ряд дополнительных компонентов, таких как Oracle Data Provider для .NET, который обеспечивает совместимость с различными версиями СУБД Oracle, и инструменты для работы с транзакциями и безопасностью.

В целом, NuGet-пакет Oracle.ManagedDataAccess упрощает работу с базой данных Oracle для .NET-разработчиков и позволяет создавать более эффективные и масштабируемые приложения, работающие с этой СУБД.

После создания подключения WPF к Oracle, была создана Entity Data Model (EDM), что позволяет легче работать с данными из базы данных.

EDM представляет собой концептуальную модель данных, которая используется для описания объектов данных и отношений между ними в базе данных. Эта модель может быть создана автоматически с помощью инструментов, предоставляемых Entity Framework, который является ORM (Object-Relational Mapping) инструментом для .NET.

Entity Framework облегчает доступ к данным в базах данных, позволяя работать с ними в терминах объектов. Это означает, что вместо написания SQL-запросов для доступа к данным, вы можете работать с объектами, которые представляют собой данные в базе данных.

Создание EDM модели позволяет WPF-приложению использовать Entity Framework для выполнения запросов к базе данных и получения данных в виде объектов, что облегчает разработку приложений и повышает их эффективность. Кроме того, EDM модель позволяет использовать легко настраиваемый подход к управлению данными, что может существенно упростить сопровождение приложения в долгосрочной перспективе.

7 Руководство пользователя

После запуска приложения запустится окно для авторизации (рисунок 11), либо, если пользователь не авторизирован, предоставляется возможность перейти по кнопке «SIGN UP NOW» на страницу для регистрации.

Рисунок 11 – Окно авторизации

При вводе имени пользователя и пароля, соответствующих главному врачу, отобразится окно с возможностями полного взаимодействия с объектами базы данных (рисунок 12).

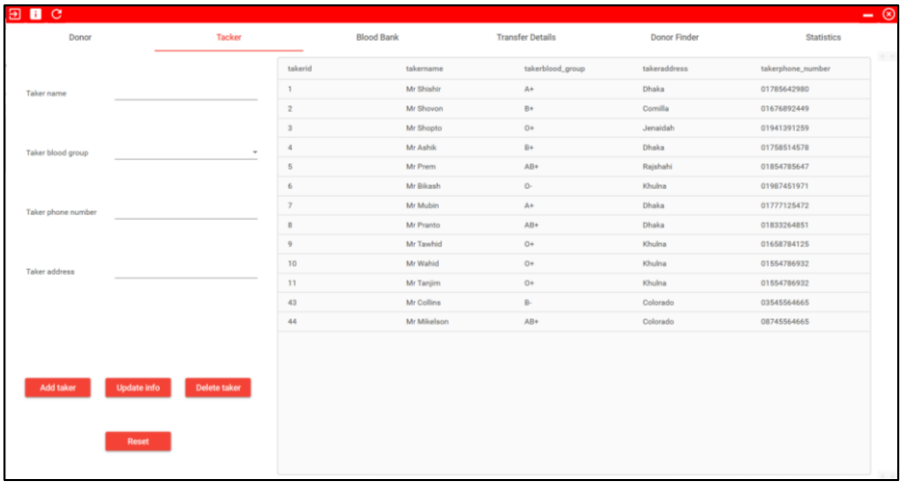

Рисунок 12 – Окно для главного врача

При авторизации под обычным пользователем-врачом, появится окно с соответствующими возможностями взаимодействия с данными базы данных.

В отличие от администратора он будет иметь право лишь просматривать данные всех таблиц, не имея возможности изменять или удалять что-то, однако за ним сохраняется возможность внесения новых доноров и реципиентов, поиск доноров по заданным параметрам и просмотр статистики.

Поиск доноров по параметрам представляет собой окно (рисунок 13) с полями для задания условий поиска. Например, когда стоит цель найти всех имеющихся доноров определенной группы крови, то в первом Combo Box выбирается группа

крови и после нажатия на кнопку «Search», в List View выведется список доноров, удовлетворяющих условию.

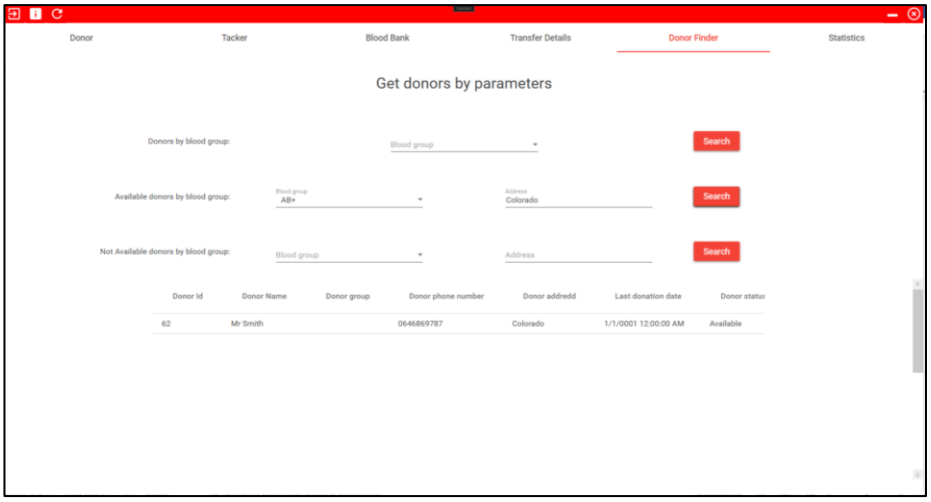

Рисунок 13 – Окно поиска доноров по параметрам

На вкладке статистики (рисунок 14) предоставляется возможность нахождения лидера по суммарному объему сданной крови донором за заданный пользователем промежуток времени.

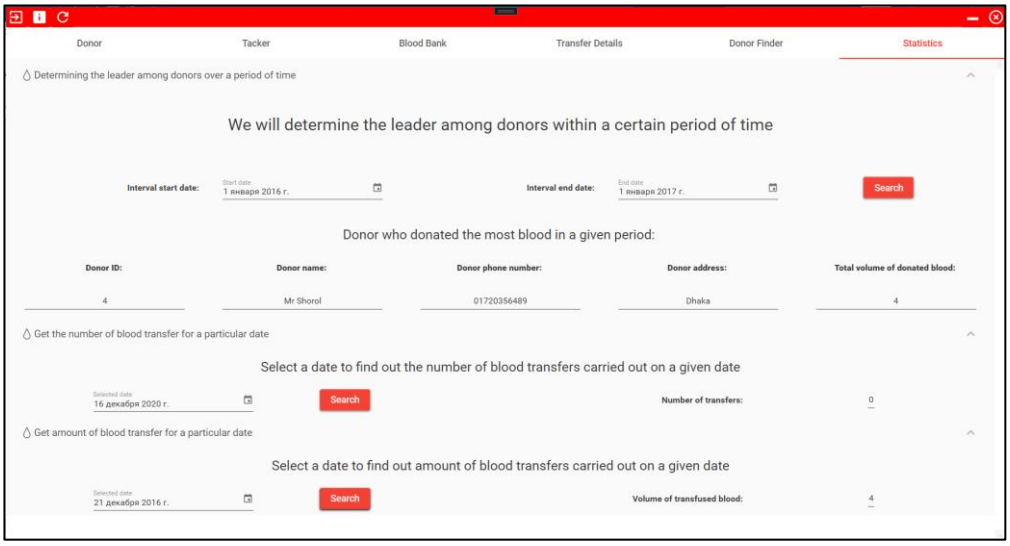

Рисунок 14 – Окно для получения некоторых статистических данных о переводах крови

А также в качестве статистических данных приложение предоставляет возможность определения числа транзакций крови и общего объема сданной крови на определенную дату.

Таким образом можно отметить, что приложение корректно и четко демонстрирует работу спроектированной в ходе курсовой работы базы данных, реализуя все заявленные функции.

Заключение

В результате выполнения курсовой работы была спроектирована база данных банка крови и с использованием ряда драйверов, для соединения базы данных со сторонним программным обеспечением, было также создано приложение для демонстрации функционирования базы данных.

В данной работе использовалось СУБД Oracle12c. И были созданы и использовались такие объекты, как таблицы, хранимые процедуры, функции, триггеры, индексы.

В связи с фактом того, что база данных «Банк крови» содержит критически важную информацию о донорах, реципиентах и банках крови, которая в случае потери или повреждения данных, может привести к непредсказуемым последствиям и серьезным проблемам, в качестве технологии была выбрана и реализована технология резервного копирования и восстановления данных.

Были созданы различные пользователи базы данных для разграничения доступа, запросы к информации исполнялись с помощью процедур.

База данных прошла тестирование при использовании большого количества данных. Также были реализованы процедуры для импорта, экспорта данных в формат XML.

В соответствии с полученным результатом работы можно сделать вывод, что разработанная база данных работает корректно, а требования технического задания выполнены в полном объёме.

Список использованных источников

1 Банк крови [Электронный ресурс]. Режим доступа: https://www.onworks.net/ru/software/app-blood-bank. Дата доступа: 28.02.2023.

2 APKCombo. Blood4Life [Электронный ресурс]. Режим доступа: https://apkcombo.com/ru/blood4life/com.mysql.test/. Дата доступа: 28.02.2023.

3 Резервное копирование и восстановление Oracle Database на виртуальной машине [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/azure/virtual-machines/workloads/oracle/oracle-database-backup-azure-storage?tabs=azure-portal. Дата доступа: 01.05.2023.

Приложение А

```
---------------------- CREATE ALL TABLES -------------------------
create table Donor(
     donor_id number generated always as identity,
     donor_name varchar(50) not null,
     donor_blood_group varchar(20) not null,
     donor_phone_number varchar(20),
     donor_address varchar(50) not null,
     last_donation_date date not null,
     donor_status varchar(20),
     primary key(donor_id)
);
create table Taker
(
     taker_id number generated always as identity,
     taker_name varchar(50) not null,
     taker_blood_group varchar(20) not null,
     taker_address varchar(50) not null,
     taker_phone_number varchar(20),
     primary key (taker_id)
);
create table Blood_Bank(
     bloodbank_id varchar(10),
     bloodbank_name varchar(50) not null,
     blood_group varchar(20)not null,
     blood_amount float,
     blood_status varchar(20),
     checking_date date,
     primary key(bloodbank_id)
);
create table Transfer_Details(
     transer_id number generated always as identity,
     bloodbank_id varchar(10),
     donor_id number,
     taker_id number,
     blood_group varchar(20),
     blood_amount float,
     transfer_date date,
     foreign key (bloodbank_id) references Blood_Bank on delete
cascade,
     foreign key (donor_id) references Donor on delete cascade,
     foreign key (taker_id) references Taker on delete cascade
);
create table Account(
     userid number generated always as identity primary key,
     login varchar2(30) not null,
     password NVARCHAR2(100)
);
create index donor_blood_group_idx on Donor (donor_blood_group);
create index taker_blood_group_idx on Taker (taker_blood_group);
create index blood_group_idx on Blood_Bank (blood_group);
create index donor_id_idx on Transfer_Details (donor_id);
```

```
create index taker_id_idx on Transfer_Details (taker_id);
create index blood_group_transfer_idx on Transfer_Details
(blood_group);
```

Приложение Б

```
ALTER SESSION SET "_ORACLE_SCRIPT"=TRUE;
--BB means bloodbank
CREATE TABLESPACE TS_BB
DATAFILE 'TS_BB.dbf'
SIZE 100m
AUTOEXTEND ON
NEXT 100m
MAXSIZE UNLIMITED
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;
--create ADMIN user
CREATE USER BB_ADMIN IDENTIFIED BY qwerty
DEFAULT TABLESPACE TS_BB
TEMPORARY TABLESPACE TEMP
QUOTA UNLIMITEd ON TS_BB;
--grant privileges to ADMIN
GRANT ALL PRIVILEGES TO BB_ADMIN;
GRANT CREATE SESSION TO BB_ADMIN;
GRANT CREATE USER TO BB_ADMIN;
GRANT CREATE ANY DIRECTORY TO BB_ADMIN WITH ADMIN OPTION;
GRANT EXECUTE ON dbms_crypto TO BB_ADMIN WITH GRANT OPTION;
GRANT EXECUTE ON dbms_xmldom TO BB_ADMIN WITH GRANT OPTION;
GRANT EXECUTE ON dbms_xslprocessor TO BB_ADMIN WITH GRANT OPTION;
GRANT EXECUTE ON dbms_xmlparser TO BB_ADMIN WITH GRANT OPTION;
GRANT EXECUTE ON dbms_xmldom TO BB_ADMIN WITH GRANT OPTION;
GRANT EXECUTE ON dbms_lob TO BB_ADMIN WITH GRANT OPTION;
--create CLIENT PROFILE
CREATE PROFILE PF_BBCLIENT LIMIT
    PASSWORD_LIFE_TIME 180
    SESSIONS_PER_USER 3
    FAILED_LOGIN_ATTEMPTS 10
    PASSWORD_LOCK_TIME 1
    PASSWORD_REUSE_TIME 10
    PASSWORD_GRACE_TIME DEFAULT
    CONNECT_TIME 180
    IDLE_TIME 30;
--create CLIENT
CREATE USER BB_CLIENT IDENTIFIED BY qwerty
DEFAULT TABLESPACE TS_BB
PROFILE PF_BBCLIENT
TEMPORARY TABLESPACE TEMP
QUOTA UNLIMITEd ON TS_BB;
--create role for CLIENT
CREATE ROLE RL_BB_CLIENT;
GRANT EXECUTE ON ADD_NEW_ACCOUNT TO RL_BB_CLIENT;
GRANT EXECUTE ON ADD_NEW_DONOR TO RL_BB_CLIENT;
GRANT EXECUTE ON ADD_NEW_TAKER TO RL_BB_CLIENT;
GRANT EXECUTE ON UPDATE_DONOR_INFO TO RL_BB_CLIENT;
GRANT EXECUTE ON UPDATE_TAKER_INFO TO RL_BB_CLIENT;
GRANT EXECUTE ON GETDONORS TO RL_BB_CLIENT;
GRANT EXECUTE ON GETTAKERS TO RL_BB_CLIENT;
```

```
GRANT EXECUTE ON GETBLOOD_BANKS TO RL_BB_CLIENT;
GRANT EXECUTE ON GETTRANSFER_DETAILS_FOR_CLIEN TO RL_BB_CLIENT;
GRANT EXECUTE ON FIND_DONORS_BY_BLOODGROUP TO RL_BB_CLIENT;
GRANT EXECUTE ON FIND_AVAIL_DONORS_BY_GR_ADDR TO RL_BB_CLIENT;
GRANT EXECUTE ON FIND_UNAVAIL_DONORS_BY_GR_ADDR TO RL_BB_CLIENT;
GRANT EXECUTE ON GET_BLOOD_LEADER TO RL_BB_CLIENT;
GRANT EXECUTE ON TOTAL_BLOOD_TRANSFER TO RL_BB_CLIENT;
GRANT EXECUTE ON NOTAL_NO TO RL_BB_CLIENT;
GRANT EXECUTE ON BLOOD_TRANSFERS_VOLUME_ON_DATE TO RL_BB_CLIENT;
commit;
--grant role to CLIENT
GRANT RL_BB_CLIENT TO BB_CLIENT;
```

Приложение В

```
------------------Trigger for Donor------------------
CREATE or REPLACE TRIGGER TR_Donor
BEFORE UPDATE OR INSERT ON Donor
FOR EACH ROW
BEGIN
      IF EXTRACT(DAY FROM(systimestamp - :new.last_donation_date)) <
90  THEN
        :new.donor_status := 'Not Available';
      else
          :new.donor_status := 'Available';
    END IF;
END TR_Donor;
------------------Trigger for Blood_Bank №1------------------
CREATE or REPLACE TRIGGER TR_Bank
BEFORE UPDATE OR INSERT ON Blood_Bank
FOR EACH ROW
BEGIN

     if :new.blood_amount < 1 then
          :new.blood_status := 'Not Available';
     elsif :new.blood_amount >1 and    :new.blood_amount <20 then
          :new.blood_status := 'Only For Emergency';
     elsif :new.blood_amount >20 and   :new.blood_amount <100 then
          :new.blood_status := 'Good Collection';
     elsif :new.blood_amount >100 then
          :new.blood_status := 'Adequate';

    END IF;
END TR_Bank;
------------------Trigger for Blood_Bank №2------------------
CREATE or REPLACE TRIGGER TR_Bank2
BEFORE UPDATE OR INSERT ON Blood_Bank
FOR EACH ROW
BEGIN

     if :new.blood_group = 'A+' or :new.blood_group = 'A-' or
:new.blood_group = 'B+' or :new.blood_group = 'B-' or
:new.blood_group = 'AB+' or :new.blood_group = 'AB-' or
:new.blood_group = 'O+' or :new.blood_group = 'O-' then
          dbms_output.put_line('Blood group insertion is Okay');
     else
          RAISE_APPLICATION_ERROR(-20000,'Incorrect Blood Group
Insertion');
    END IF;
END TR_Bank2;
```

Приложение Г

```
----------------------SELECT without parameters-------------------
--1. Get all donors
CREATE OR REPLACE PROCEDURE GETDONORS(clients OUT SYS_REFCURSOR) IS
BEGIN
    OPEN clients FOR
    SELECT donor_id, donor_name, donor_blood_group,
donor_phone_number, donor_address, last_donation_date, donor_status
FROM Donor;
END;
--2. Get all takers
CREATE OR REPLACE PROCEDURE GETTAKERS(takers OUT SYS_REFCURSOR) IS
BEGIN
    OPEN takers FOR
    SELECT taker_id, taker_name, taker_blood_group, taker_address,
taker_phone_number FROM Taker;
END;
--3. Get all blood_banks
CREATE OR REPLACE PROCEDURE GETBLOOD_BANKS(blood_banks OUT
SYS_REFCURSOR) IS
BEGIN
    OPEN blood_banks FOR
    SELECT bloodbank_id, bloodbank_name, blood_group, blood_amount,
blood_status, checking_date from blood_bank; END;
--4. Get all transfer details for admin
CREATE OR REPLACE PROCEDURE GETTRANSFER_DETAILS(
    transfer_details OUT SYS_REFCURSOR
) IS BEGIN
    OPEN transfer_details FOR
    SELECT transer_id, bloodbank_id, donor_id, taker_id, blood_group,
blood_amount, transfer_date FROM transfer_details; END;
--5. Get all transfer details for client
CREATE OR REPLACE PROCEDURE GETTRANSFER_DETAILS_FOR_CLIEN(
    transfer_details OUT SYS_REFCURSOR
) IS BEGIN
    OPEN transfer_details FOR
    SELECT tr.transer_id, bb.bloodbank_name, d.donor_name,
t.taker_name, tr.blood_group, tr.blood_amount, tr.transfer_date
    FROM Transfer_details tr
    JOIN Donor d on d.donor_id = tr.donor_id
    JOIN Taker t ON t.taker_id = tr.taker_id
    JOIN Blood_bank bb on bb.bloodbank_id = tr.bloodbank_id;
END;
--6. Get all accounts details for admin
CREATE OR REPLACE PROCEDURE GETACCOUNTS(accunts OUT SYS_REFCURSOR) IS
BEGIN
    OPEN accunts FOR
    SELECT userid, login, password FROM account; END;
```

Приложение Д

```
--------------------SELECT with parameters--------------------
-------1. Get donors by bloodgroup
CREATE OR REPLACE PROCEDURE
FIND_DONORS_BY_BLOODGROUP(check_blood_group IN VARCHAR, donor_cursor
OUT SYS_REFCURSOR) IS
BEGIN
    OPEN donor_cursor FOR
    SELECT donor_id, donor_name, donor_phone_number, donor_address,
last_donation_date, donor_status
    FROM Donor
    WHERE donor_blood_group = check_blood_group;
END;
-------2. Get available donors by donor_blood_group and donor_adress
CREATE OR REPLACE PROCEDURE FIND_AVAIL_DONORS_BY_GR_ADDR(
  check_blood_group IN VARCHAR,
  check_donoe_address IN VARCHAR,
  donor_cursor OUT SYS_REFCURSOR
) IS
BEGIN
    OPEN donor_cursor FOR
    SELECT donor_id, donor_name, donor_phone_number, donor_address,
donor_status
    FROM Donor
    WHERE donor_blood_group = check_blood_group
          AND donor_address = check_donoe_address
          AND donor_status = 'Available';
END;
-------3. Get unavailable donors by donor_blood_group and
donor_adress
CREATE OR REPLACE PROCEDURE FIND_UNAVAIL_DONORS_BY_GR_ADDR(
  check_blood_group IN VARCHAR,
  check_donoe_address IN VARCHAR,
  donor_cursor OUT SYS_REFCURSOR
) IS
BEGIN
    OPEN donor_cursor FOR
    SELECT donor_id, donor_name, donor_phone_number, donor_address,
donor_status
    FROM Donor
    WHERE donor_blood_group = check_blood_group
          AND donor_address = check_donoe_address
          AND donor_status = 'Not Available';
END;
------------------------------------------------------------------
---------Determination of the leader among donors within a period of
time specified by the user
CREATE OR REPLACE PROCEDURE GET_BLOOD_LEADER(
    start_date IN DATE,
    end_date IN DATE,
    blood_leader_cursor OUT SYS_REFCURSOR
) IS
```

```
BEGIN
    OPEN blood_leader_cursor FOR
      SELECT d.donor_id, d.donor_name, d.donor_phone_number,
d.donor_address, SUM(td.blood_amount) AS total_blood_amount
      FROM Donor d
      JOIN Transfer_Details td ON d.donor_id = td.donor_id
        WHERE td.transfer_date BETWEEN start_date AND end_date
      GROUP BY d.donor_id, d.donor_name, d.donor_phone_number,
d.donor_address, d.donor_blood_group
      ORDER BY total_blood_amount DESC
    FETCH FIRST ROW ONLY;
END;
```

Приложение Е

```
--------------INSERT-------------------
----1. Add new donor -----
CREATE OR REPLACE PROCEDURE ADD_NEW_DONOR(
  dname Donor.donor_name%TYPE,
  dblood Donor.donor_blood_group%TYPE,
  daddress Donor.donor_address%TYPE,
  dcontact Donor.donor_phone_number%TYPE,
  dldd Donor.last_donation_date%TYPE
  ) IS
BEGIN
  insert into
Donor(donor_name,donor_blood_group,donor_address,donor_phone_number,l
ast_donation_date,donor_status)
  values (dname,dblood,daddress,dcontact,dldd,null);
  commit;
END ADD_NEW_DONOR;
----2. Add new taker -----
CREATE OR REPLACE PROCEDURE ADD_NEW_TAKER(
  tname Taker.taker_name%TYPE,
  tblood Taker.taker_blood_group%TYPE,
  taddress Taker.taker_address%TYPE,
  tcontact Taker.taker_phone_number%TYPE
  ) IS
BEGIN
  insert into
Taker(taker_name,taker_blood_group,taker_address,taker_phone_number)
  values (tname,tblood,taddress,tcontact);
  commit;
END ADD_NEW_TAKER;
----3. Add new blood bank-----
CREATE OR REPLACE PROCEDURE ADD_NEW_BLOOD_BANK(
  bbid Blood_Bank.bloodbank_id%TYPE,
  bbname Blood_Bank.bloodbank_name%TYPE,
  bbbgroup Blood_Bank.blood_group%TYPE,
  bbbamount Blood_Bank.blood_amount%TYPE,
  bbchaeckingd Blood_Bank.checking_date%TYPE
  ) IS
BEGIN
  insert into
Blood_Bank(bloodbank_id,bloodbank_name,blood_group,blood_amount,blood
_status,checking_date)
  values (bbid,bbname,bbbgroup,bbbamount,null,bbchaeckingd);
  commit;
END ADD_NEW_BLOOD_BANK;
----4. Add new Transfer-----
CREATE OR REPLACE PROCEDURE ADD_NEW_TRANSFER_DETAILS(
  bbid transfer_details.bloodbank_id%TYPE,
  donorid transfer_details.donor_id%TYPE,
  takerid transfer_details.taker_id%TYPE,
  bloodgroup transfer_details.blood_group%TYPE,
  bloodamount transfer_details.blood_amount%TYPE,
```

```
  transferdate transfer_details.transfer_date%TYPE
  ) IS
BEGIN
  insert into
transfer_details(bloodbank_id,donor_id,taker_id,blood_group,blood_amo
unt,transfer_date)
  values (bbid,donorid,takerid,bloodgroup,bloodamount,transferdate);
  commit;
END ADD_NEW_TRANSFER_DETAILS;
----5. Add new Account-----
CREATE OR REPLACE PROCEDURE ADD_NEW_ACCOUNT(
    p_login account.login%TYPE,
    p_password account.password%TYPE) IS
BEGIN
    INSERT INTO Account(login, password)
    VALUES(p_login, p_password);
    COMMIT;
END;
---------------UPDATE--------------------
----1. Update donor -----
CREATE OR REPLACE PROCEDURE UPDATE_DONOR_INFO(
  d_id Donor.donor_id%TYPE,
  dname Donor.donor_name%TYPE,
  daddress Donor.donor_address%TYPE,
  dcontact Donor.donor_phone_number%TYPE,
  dldd Donor.last_donation_date%TYPE
) IS
BEGIN
  UPDATE Donor
  SET donor_name = dname,
  donor_address = daddress,
  donor_phone_number = dcontact,
  last_donation_date = dldd
  WHERE donor_id = d_id;
  COMMIT;
END UPDATE_DONOR_INFO;
----2. Update taker -----
CREATE OR REPLACE PROCEDURE UPDATE_TAKER_INFO(
  t_id Taker.taker_id%TYPE,
  tname Taker.taker_name%TYPE,
  taddress Taker.taker_address%TYPE,
  tcontact Taker.taker_phone_number%TYPE
) IS
BEGIN
  UPDATE Taker
    SET taker_name = tname,
    taker_address = taddress,
    taker_phone_number = tcontact
    WHERE taker_id = t_id;
  COMMIT;
END UPDATE_TAKER_INFO;
----3. Update blood bank -----
CREATE OR REPLACE PROCEDURE UPDATE_BLOOD_BANK(
```

```
     id Blood_Bank.bloodbank_id%TYPE,
     bgroup Blood_Bank.blood_group%TYPE,
     amount Blood_Bank.blood_amount%TYPE
) IS
BEGIN
     UPDATE Blood_Bank set
blood_amount=amount,checking_date=current_date where bloodbank_id=id
and blood_group=bgroup;
EXCEPTION
  WHEN no_data_found THEN
    RAISE_APPLICATION_ERROR(-20203, 'No Data found.');
    COMMIT;
END UPDATE_BLOOD_BANK;
------4. Update transfer_details
CREATE OR REPLACE PROCEDURE UPDATE_TRANSFER_DETAILS(
  tid Transfer_Details.transer_id%TYPE,
  bbid Transfer_Details.bloodbank_id%TYPE,
  donorid Transfer_Details.donor_id%TYPE,
  takerid Transfer_Details.taker_id%TYPE,
  bloodgroup Transfer_Details.blood_group%TYPE,
  bloodamount Transfer_Details.blood_amount%TYPE,
  transferdate Transfer_Details.transfer_date%TYPE
) IS
BEGIN
  UPDATE Transfer_Details
  SET bloodbank_id = bbid,
      donor_id = donorid,
      taker_id = takerid,
      blood_group = bloodgroup,
      blood_amount = bloodamount,
      transfer_date = transferdate
  WHERE transer_id = tid;
  COMMIT;
END UPDATE_TRANSFER_DETAILS;
-----5. Update Account
CREATE OR REPLACE PROCEDURE UPDATE_ACCOUNT(
  accid account.userid%TYPE,
  acclogin account.login%TYPE,
  accpassword account.password%TYPE
) IS
BEGIN
  UPDATE Account set login=acclogin,password=accpassword where
userid=accid;
EXCEPTION
  WHEN no_data_found THEN
    RAISE_APPLICATION_ERROR(-20203, 'No Data found.');
  COMMIT;
END UPDATE_ACCOUNT;
---------------DELETE--------------------
----1. Delete Donor
CREATE OR REPLACE PROCEDURE DELETE_DONOR(
  d_id donor.donor_id%TYPE
) IS
```

```
BEGIN
  DELETE FROM Donor WHERE donor_id = d_id;
  COMMIT;
END DELETE_DONOR;
----2. Delete Taker
CREATE OR REPLACE PROCEDURE DELETE_TAKER(
  t_id Taker.taker_id%TYPE
) IS
BEGIN
  DELETE FROM Taker WHERE taker_id = t_id;
  COMMIT;
END DELETE_TAKER;
---4. Delete Transfer Details
CREATE OR REPLACE PROCEDURE DELETE_TRANSFER_DETAILS(
  transferid transfer_details.transer_id%TYPE
) IS
BEGIN
  DELETE FROM Transfer_Details WHERE transer_id = transferid;
  COMMIT;
END DELETE_TRANSFER_DETAILS;
-----5. Delete Account
CREATE OR REPLACE PROCEDURE DELETE_ACCOUNT(
  accountid account.userid%TYPE
) IS
BEGIN
  DELETE FROM Account WHERE userid = accountid;
  COMMIT;
END DELETE_ACCOUNT;
```

Приложение Ж

```
--1. Using Function to show the amount of blood transfer for a
particular blood group --
CREATE OR REPLACE FUNCTION TOTAL_BLOOD_TRANSFER (p_blood_group IN
VARCHAR) RETURN NUMBER
IS
     total_t NUMBER(4);
BEGIN
    SELECT sum(blood_amount) into total_t from Blood_Bank where
blood_group=p_blood_group;
    RETURN total_t;
END TOTAL_BLOOD_TRANSFER;
--2. Using Function to show the number of blood transfer for a
particular date --
CREATE OR REPLACE FUNCTION NOTAL_NO(transfer_date_in IN DATE) RETURN
NUMBER is
  cnt NUMBER(4);
BEGIN
    SELECT count(bloodbank_id) into cnt from Transfer_Details where
transfer_date = transfer_date_in;
    RETURN cnt;
END NOTAL_NO;
--3. A function that determines amount of blood transfer on a
specific date-
CREATE OR REPLACE FUNCTION
BLOOD_TRANSFERS_VOLUME_ON_DATE(transfer_date IN DATE) RETURN FLOAT IS
  total_volume FLOAT;
BEGIN
  SELECT SUM(blood_amount) INTO total_volume FROM Transfer_Details
WHERE transfer_date = transfer_date;
  RETURN total_volume;
END BLOOD_TRANSFERS_VOLUME_ON_DATE;
```

Приложение 3

```
----------------------IMPORT-------------------------------
SELECT file_name FROM dba_data_files;
CREATE OR REPLACE DIRECTORY EXPORT_DATA AS
'C:/TakerExportImport/xml/export';
CREATE OR REPLACE DIRECTORY IMPORT_DATA AS
'C:/TakerExportImport/xml/import';
CREATE OR REPLACE PROCEDURE EXPORT_TAKERS_TO_XML
IS
    doc DBMS_XMLDOM.DOMDocument;
    xdata XMLTYPE;
    CURSOR xmlcur IS
        SELECT XMLELEMENT(
          "Takers",
           XMLAttributes('http://www.w3.org/2001/XMLSchema' AS
"xmlns:xsi",
            'http://www.oracle.com/Users.xsd' AS
"xsi:nonamespaceSchemaLocation"),
              XMLAGG(XMLELEMENT("Taker",
              xmlelement("id", Taker.taker_id),
              xmlelement("name", Taker.taker_name),
              xmlelement("blood_group", Taker.taker_blood_group),
              xmlelement("address", Taker.taker_address),
              xmlelement("phone_number", Taker.taker_phone_number)
            ))) from Taker;
BEGIN
open xmlcur;
    LOOP
        FETCH xmlcur INTO xdata;
        EXIT WHEN xmlcur%notfound;
    END LOOP;
    CLOSE xmlcur;
    doc := DBMS_XMLDOM.NewDOMDocument(xdata);
    DBMS_XMLDOM.WRITETOFILE(doc, 'EXPORT_DATA/Takers.xml');
END;
----------------------IMPORT-------------------------------
CREATE OR REPLACE PROCEDURE IMPORT_TAKERS_FROM_XML
IS
    L_CLOB CLOB;
    L_BFILE BFILE := BFILENAME('IMPORT_DATA', 'takers.xml');
    L_DEST_OFFSET INTEGER := 1;
    L_SRC_OFFSET INTEGER := 1;
    L_BFILE_CSID NUMBER := 0;
    L_LANG_CONTEXT INTEGER := 0;
    L_WARNING INTEGER := 0;
    P DBMS_XMLPARSER.PARSER;
    V_DOC DBMS_XMLDOM.DOMDOCUMENT;
    V_ROOT_ELEMENT DBMS_XMLDOM.DOMELEMENT;
    V_CHILD_NODES DBMS_XMLDOM.DOMNODELIST;
    V_CURRENT_NODE DBMS_XMLDOM.DOMNODE;
    tk Taker%ROWTYPE;
BEGIN
```

```
    DBMS_LOB.CREATETEMPORARY (L_CLOB, TRUE);
    DBMS_LOB.FILEOPEN(L_BFILE, DBMS_LOB.FILE_READONLY);
    DBMS_LOB.LOADCLOBFROMFILE(DEST_LOB => L_CLOB, SRC_BFILE =>
L_BFILE, AMOUNT => DBMS_LOB.LOBMAXSIZE,
        DEST_OFFSET => L_DEST_OFFSET, SRC_OFFSET => L_SRC_OFFSET,
BFILE_CSID => L_BFILE_CSID,
        LANG_CONTEXT => L_LANG_CONTEXT, WARNING => L_WARNING);
    DBMS_LOB.FILECLOSE(L_BFILE);
    COMMIT;
    P := DBMS_XMLPARSER.NEWPARSER;
    DBMS_XMLPARSER.PARSECLOB(P, L_CLOB);
    V_DOC := DBMS_XMLPARSER.GETDOCUMENT(P);
    V_ROOT_ELEMENT := DBMS_XMLDOM.Getdocumentelement(V_DOC);
    V_CHILD_NODES := DBMS_XMLDOM.GETCHILDRENBYTAGNAME(V_ROOT_ELEMENT,
'*');
    FOR i IN 0 .. DBMS_XMLDOM.GETLENGTH(V_CHILD_NODES) - 1 LOOP
        V_CURRENT_NODE := DBMS_XMLDOM.ITEM(V_CHILD_NODES, i);
        DBMS_XSLPROCESSOR.VALUEOF(V_CURRENT_NODE,
            'name/text()', tk.taker_name);
        DBMS_XSLPROCESSOR.VALUEOF(V_CURRENT_NODE,
            'blood_group/text()', tk.taker_blood_group);
        DBMS_XSLPROCESSOR.VALUEOF(V_CURRENT_NODE,
            'address/text()', tk.taker_address);
        DBMS_XSLPROCESSOR.VALUEOF(V_CURRENT_NODE,
            'phone_number/text()', tk.taker_phone_number);
        INSERT INTO Taker (taker_name, taker_blood_group,
taker_address, taker_phone_number)
        VALUES (tk.taker_name, tk.taker_blood_group,
tk.taker_address, tk.taker_phone_number);
END LOOP;

    DBMS_LOB.FREETEMPORARY(L_CLOB);
    DBMS_XMLPARSER.FREEPARSER(P);
    DBMS_XMLDOM.FREEDOCUMENT(V_DOC);
    COMMIT;
END;
```

Приложение И

```
> mkdir C:\BackupBB\
> sqlplus / as sysdba
SQL> startup
RMAN> connect target /
RMAN> alter system set db_recovery_file_dest_size=4096M scope=both;
RMAN> alter system set db_recovery_file_dest=' C:\BackupBB\'
scope=both;
SQL> SELECT log_mode FROM v$database;
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE ARCHIVELOG;
SQL> ALTER DATABASE OPEN;
SQL> ALTER SYSTEM SWITCH LOGFILE;
RMAN> backup as compressed backupset database plus archivelog;
Удаление файлов dbf
> rman
RMAN> connect target /
RMAN> shutdown abort
RMAN> startup mount
RMAN> restore database;
RMAN> recover database;
RMAN> alter database open;
```