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 5](#_Toc134772080)

[1. Аналитический обзор литературы 6](#_Toc134772081)

[1.1 Изучение требований, определение вариантов использования 6](#_Toc134772082)

[1.2 Анализ аналогичных решений 6](#_Toc134772083)

[1.3 Вывод по разделу 8](#_Toc134772084)

[2. Проектирование базы данных 9](#_Toc134772085)

[2.1 Разработка модели базы данных 9](#_Toc134772086)

[2.2 Описание информационных объектов и ограничений целостности 10](#_Toc134772087)

[2.2.1 Таблица донора 10](#_Toc134772088)

[2.2.2 Таблица получателя крови 10](#_Toc134772089)

[2.2.4 Таблица банка крови 11](#_Toc134772090)

[2.2.5 Таблица транзакции крови 11](#_Toc134772091)

[2.2.6 Таблица аккаунта пользователя 11](#_Toc134772092)

[2.3 Разработка объектов базы данных 13](#_Toc134772093)

[2.3.1 Пользователи 13](#_Toc134772094)

[2.3.2 Триггеры базы данных 13](#_Toc134772095)

[2.3.3 Процедуры базы данных 14](#_Toc134772096)

[2.3.4 Функции базы данных 15](#_Toc134772097)

[3. Описание процедур импорта и экспорта 16](#_Toc134772098)

[4. Тестирование производительности 18](#_Toc134772099)

[5. Описание технологии 20](#_Toc134772100)

[6. Краткое описание приложения для демонстрации 23](#_Toc134772101)

[7. Руководство пользователя 24](#_Toc134772102)

[Заключение 26](#_Toc134772103)

[Список использованных источников 27](#_Toc134772104)

[Приложение А 28](#_Toc134772105)

[Приложение Б 29](#_Toc134772106)

[Приложение В 31](#_Toc134772107)

[Приложение Г 32](#_Toc134772108)

[Приложение Д 33](#_Toc134772109)

[Приложение Е 35](#_Toc134772110)

[Приложение Ж 39](#_Toc134772111)

[Приложение З 40](#_Toc134772112)

[Приложение И 42](#_Toc134772113)

**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.

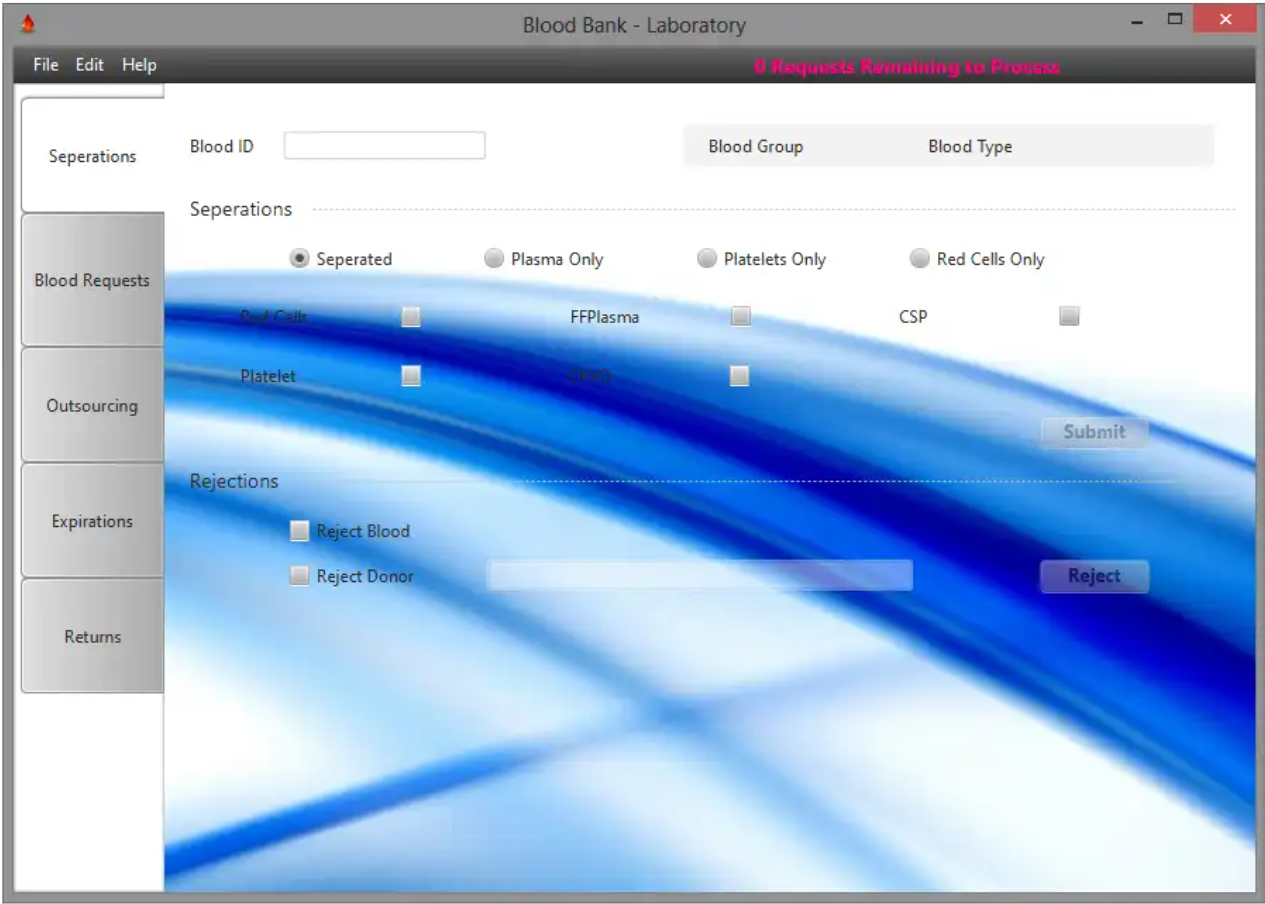


Рисунок 1 – Скриншот интерфейса приложения «Blood Bank»

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.

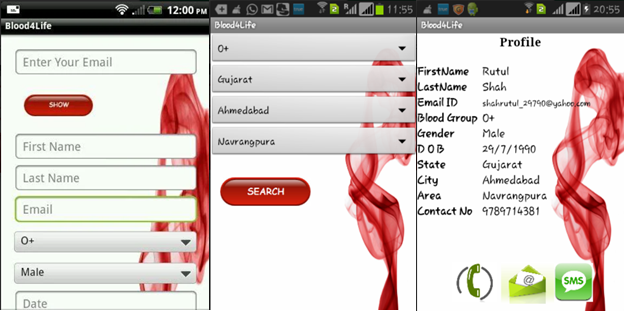


Рисунок 2 – Скриншот интерфейса приложения «Blood4Life»

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.

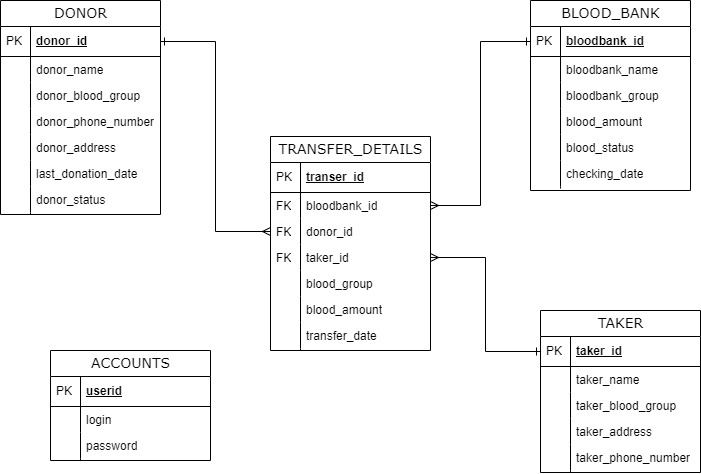


Рисунок 3 – Диаграмма базы данных

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 З).

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.

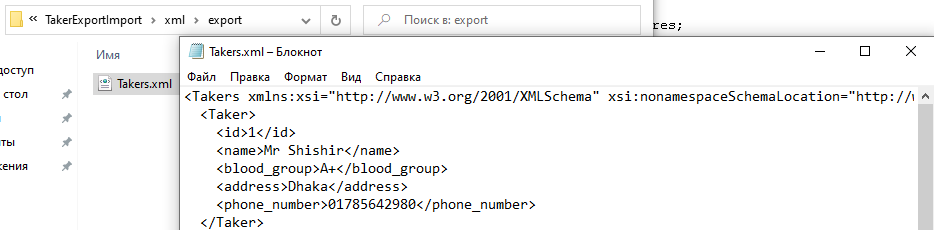


Рисунок 4 – Результат выполнения процедуры EXPORT\_TAKERS\_TO\_XML

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».

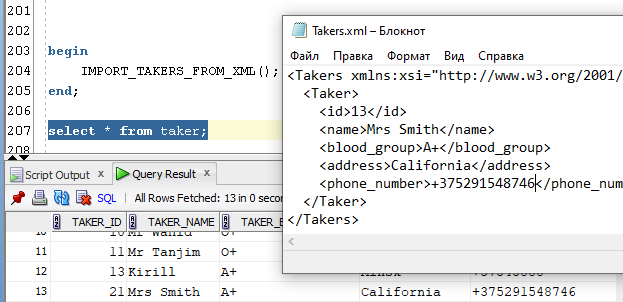


Рисунок 5 – Результат выполнения процедуры IMPORT\_TAKERS\_TO\_XML

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

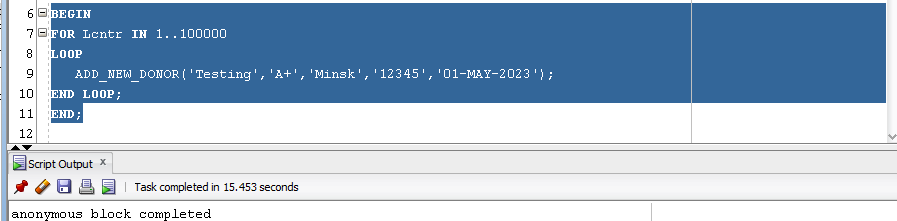


Рисунок 6 – Добавление 100000 строк в таблицу Donors.

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.

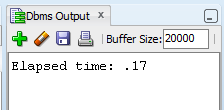


Рисунок 7 – Время выполнения анонимного блока.

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.

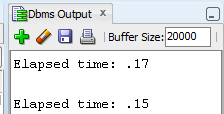


Рисунок 8 – Время выполнения анонимного блока после создания индекса.

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.

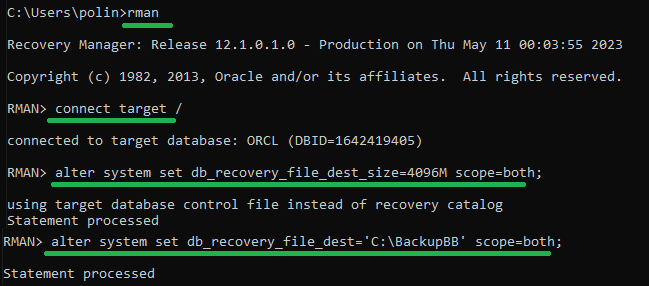


Рисунок 9 – Задание переменных среды базы данных для области быстрого восстановления при помощи утилиты RMAN.

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

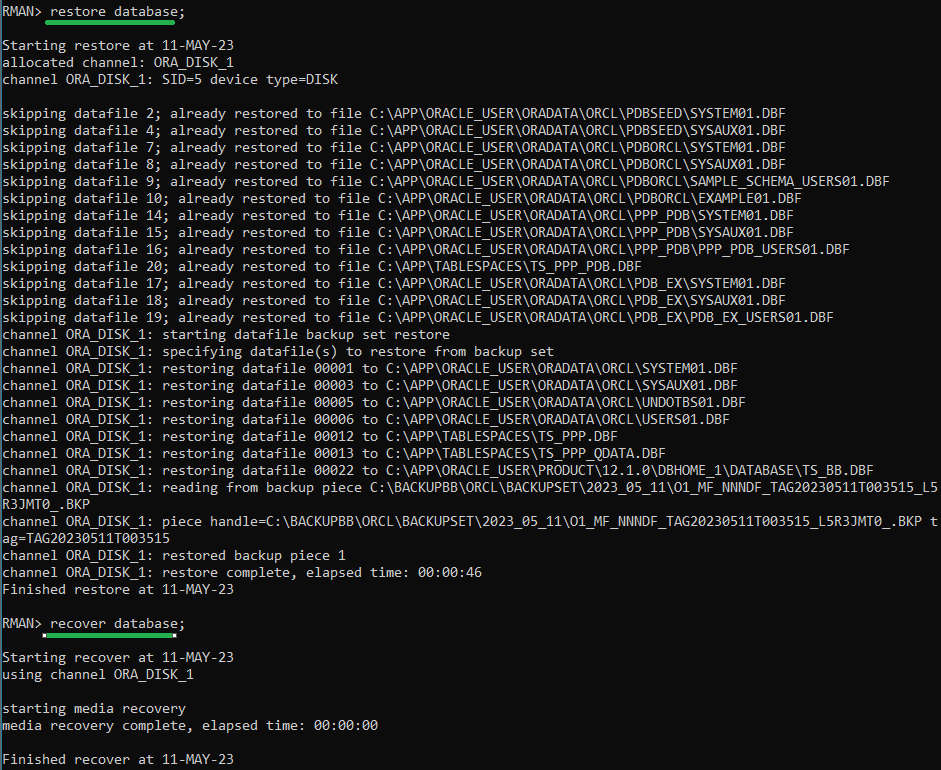


Рисунок 10 – Результат выполнения команд восстановления данных при помощи утилиты RMAN

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.

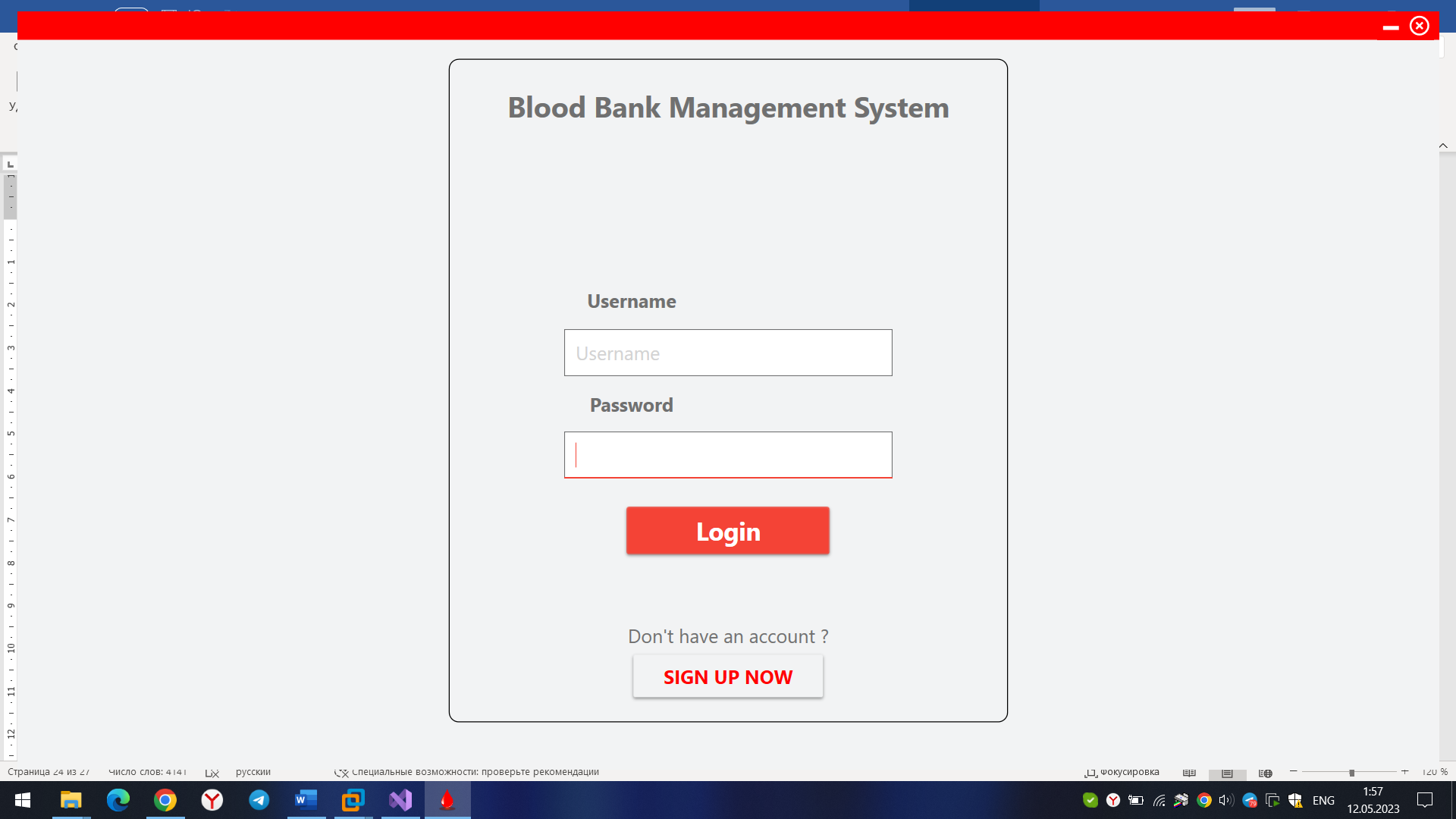


Рисунок 11 – Окно авторизации

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

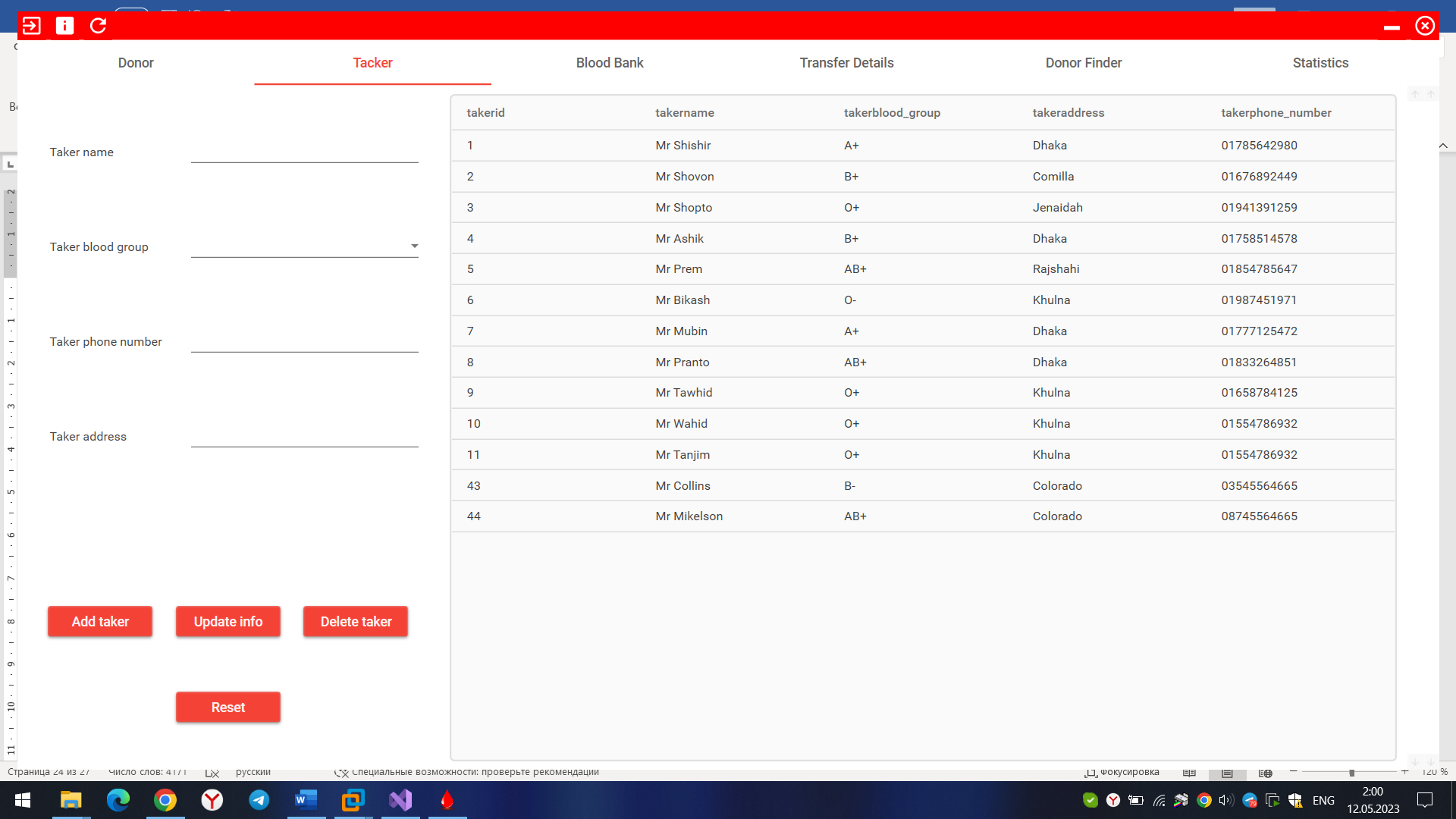


Рисунок 12 – Окно для главного врача

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.

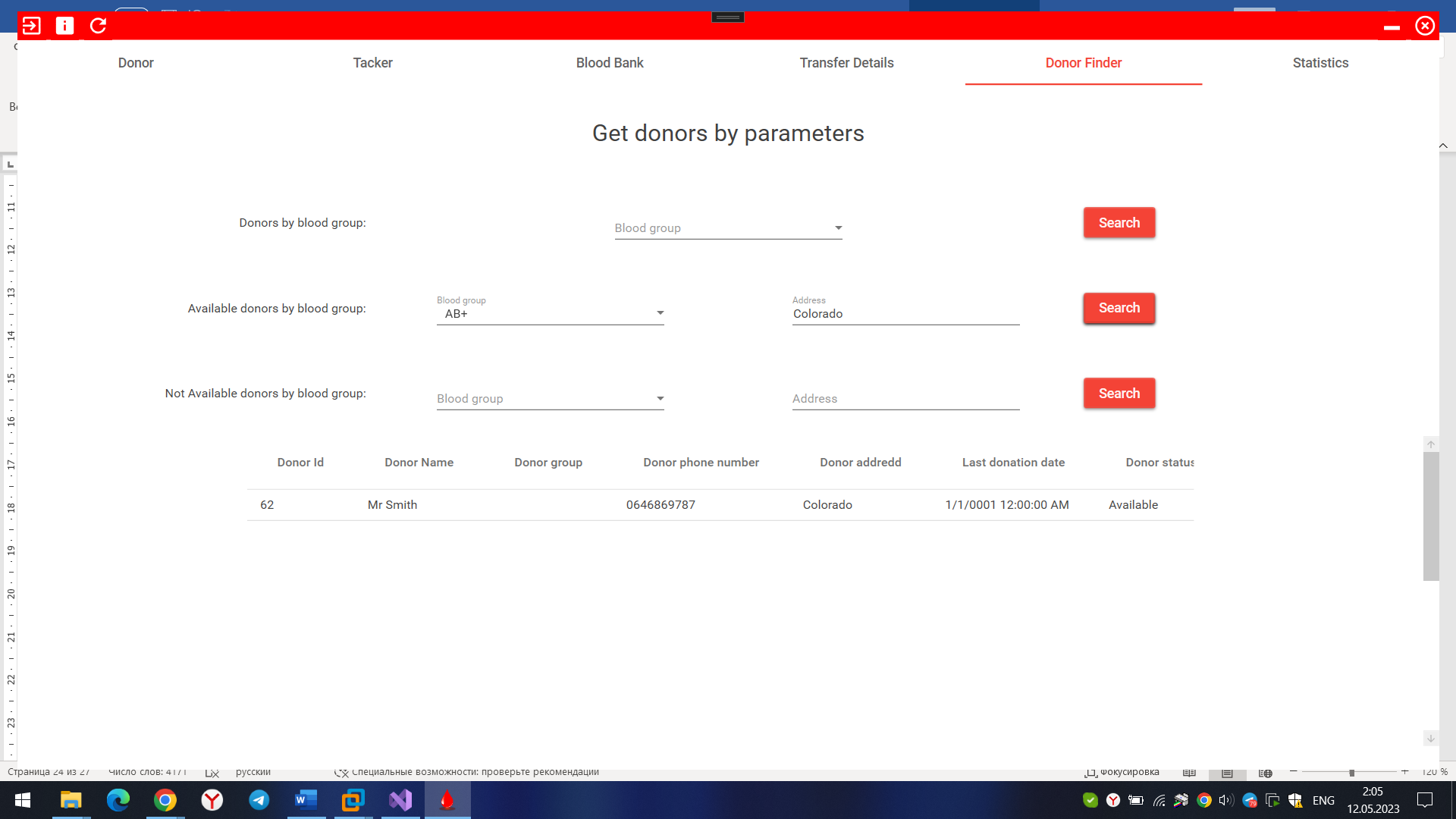


Рисунок 13 – Окно поиска доноров по параметрам

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

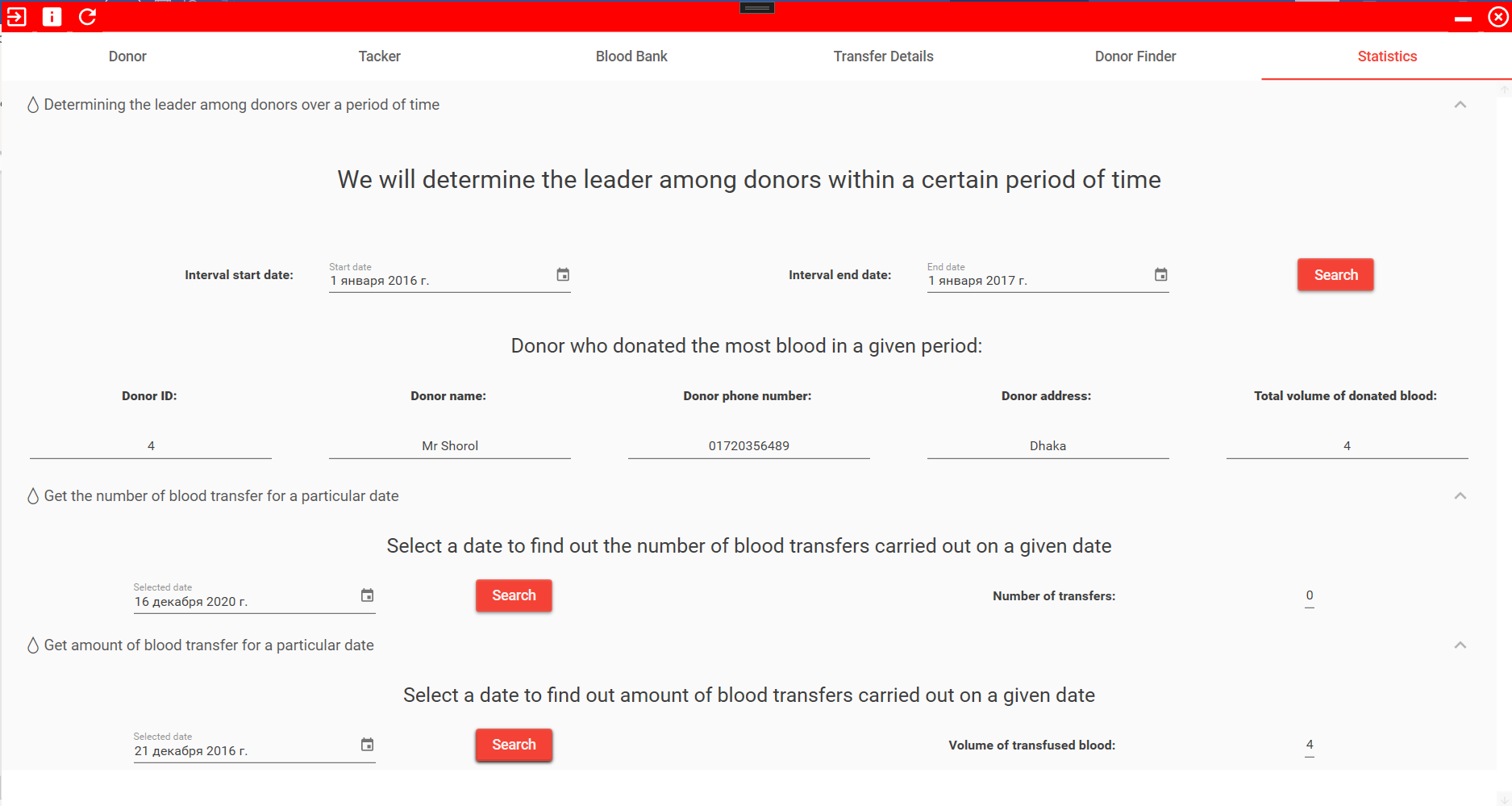


Рисунок 14 – Окно для получения некоторых статистических данных о переводах крови

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,last\_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\_amount,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 З**

-----------------------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;