

Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Базы данных

Тема «Репетиционная база»
Лабораторная работа №6
Создание приложения для базы данных

Студент:
Преподаватель:

А.С. Бригадир
Д.В. Куприянова

МИНСК 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	4
1.1 Запуск приложения.....	4
1.2 Интерфейс приложения.....	4
1.3 Добавление и удаление таблиц	6
1.4 Добавление, редактирование и удаление строк	6
1.4.1 Добавление строк	6
1.4.2 Редактирование строк.....	7
1.4.3 Удаление строк.....	7
1.5 Вывод результатов запросов.....	7
1.5.1 Вывод результатов существующих запросов	7
1.5.2 Сохранение новых запросов	7
1.6 Создание резервной копии.....	8
1.7 Механизм экспорта в файл Excel	9
2 ЛИСТИНГ КОДА.....	10
ЗАКЛЮЧЕНИЕ	60

ВВЕДЕНИЕ

Главной целью работы является создание удобного инструмента для работы с базой данных, который позволяет пользователю выполнять следующие задачи:

- добавление новых таблиц в базу данных;
- удаление существующих таблиц;
- работа с данными в таблицах: добавление, редактирование и удаление строк;
- создание резервных копий для восстановления удаленных таблиц, строк и всей базы данных;
- выполнение SQL-запросов с возможностью сохранения для последующего использования;
- выполнение SQL-запросов из 4 и 5 лабораторной работы;
- экспорт данных (таблиц, результатов запросов, всей базы данных) в файл формата Excel.

Для разработки данного приложения был выбран язык программирования C#, для создания графического интерфейса платформа WPF (Windows Presentation Foundation). В качестве архитектуры была выбрана модель MVVM (Model-View-ViewModel) для разделения логики приложения и интерфейса.

1 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

1.1 Запуск приложения

Системные требования: операционная система Windows 10 и выше, .NET Framework 4.7.2, PostgreSQL. Для начала работы необходимо запустить файл RepBase.exe в папке Release со всеми зависимостями и конфигурационными файлами.

1.2 Интерфейс приложения

Главное окно приложения состоит из двух глобальных областей: слева находится панель управления, посередине область для отображения данных. Главное окно приложения приведено на рисунке 1.1.

id	name	airconditioner	price	recording_support	area	rehearsal_point
2	Medium Room	✓	40	✓	25	33
3	Small Room	✓	30	✓	15	33
6	Green Room	✓	25	✓	28	34
7	Large Room	✓	30	✓	35	35
8	Medium Room	✓	45	✓	25	35
9	Small Room	✓	20	✓	12	35
10	Yellow Room	✓	30	✓	30	36
11	Orange Room	✓	35	✓	18	36
13	Large Room	✓	50	✓	35	37
14	Medium Room	✓	30	✓	28	37
15	Small Room	✓	15	✓	14	37
16	Black Room	✓	45	✓	29	38
17	White Room	✓	25	✓	19	38
18	Gray Room	✓	20	✓	27	38
19	Large Room	✓	50	✓	35	39
20	Medium Room	✓	40	✓	24	39
21	Small Room	✓	30	✓	13	39
22	Cyan Room	✓	35	✓	31	40
23	Magenta Room	✓	30	✓	21	40
24	Brown Room	✓	45	✓	23	40
25	Large Room	✓	50	✓	35	41
26	Medium Room	✓	40	✓	27	41
27	Small Room	✓	15	✓	11	41
28	Light Blue Room	✓	25	✓	30	42
29	Light Green Room	✓	20	✓	20	42
30	Dark Room	✓	30	✓	18	42
31	Meeting Room	✓	35	✓	22	43
32	Rehearsal Room	✓	10	✓	14	43
1	Large Room	✓	50	✓	35	43
4	Blue Room	✓	45	✓	30	44
33	Big Room	✓	25	✓	20	43
12	Purple Room	✓	40	✓	22	46
34	Small Room	✓	30	✓	15	48
5	Red Room	✓	35	✓	20	44
35	Small Room	✓	15	✓	15	44

Рисунок 1.1 – Главное окно приложения

Панель управления включает несколько областей, объединенных по общему смыслу и назначению. Рассмотрим каждую область подробно.

Область управления таблицами содержит список таблиц и кнопки для создания новых и удаления существующих таблиц. Данная область приведена на рисунке 1.2.

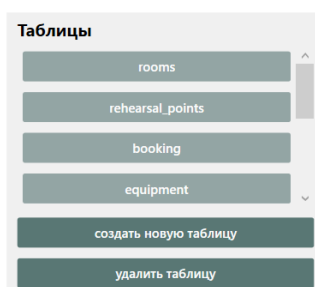


Рисунок 1.2 – Область управления таблицами

Область управления содержанием таблицы содержит кнопки для добавления и сохранения в бд новой строки, а также кнопку для удаления существующей строки. Данная область приведена на рисунке 1.3.

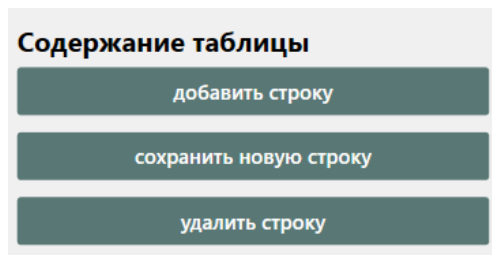


Рисунок 1.3 – Область управления содержанием таблицы

Область управления скриптами включает комбобокс, который содержит названия изначально заданных и добавленных пользователем в процессе работы скриптов, поле для ввода и отображения скрипта, кнопки для выполнения и сохранения нового скрипта в комбобоксе. Данная область приведена на рисунке 1.4.

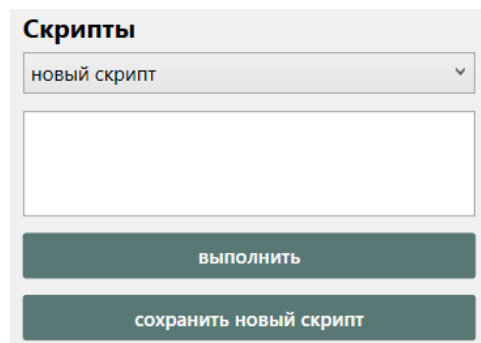


Рисунок 1.4 – Область управления скриптами

Область резервного сохранения включает кнопки для создания бэкапа базы данных, восстановления состояния базы данных из созданных бэкапов, а также кнопка для экспорта данных в файл Excel. Данная область приведена на рисунке 1.5.

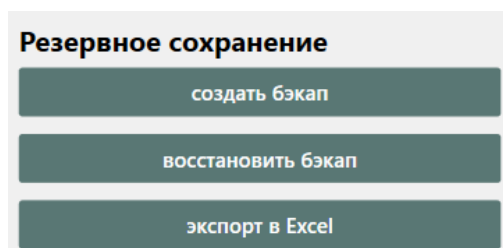


Рисунок 1.5 – Область резервного сохранения

1.3 Добавление и удаление таблиц

Для добавления новой таблицы необходимо нажать на кнопку «Создать новую таблицу» в области управления таблицами. Будет вызвано окно создания таблицы, отображенное на рисунке 1.6. Необходимо ввести название таблицы в соответствующее поле, после этого нажатием кнопки «Добавить столбец» добавить в таблицу необходимое количество столбцов. Для задания имени столбца необходимо ввести его в поле ввода в столбце «Название столбца». Для задания типа столбца необходимо выбрать его в выпадающем списке. Для удаления лишних столбцов необходимо нажать на кнопку «Удалить» рядом с соответствующим столбцом. Для закрытия окна без создания таблицы необходимо нажать на кнопку «Выход». Для создания таблицы нажать на кнопку «Создать таблицу». Столбец с id будет сгенерирован автоматически в том случае, если пользователь не добавил его сам.

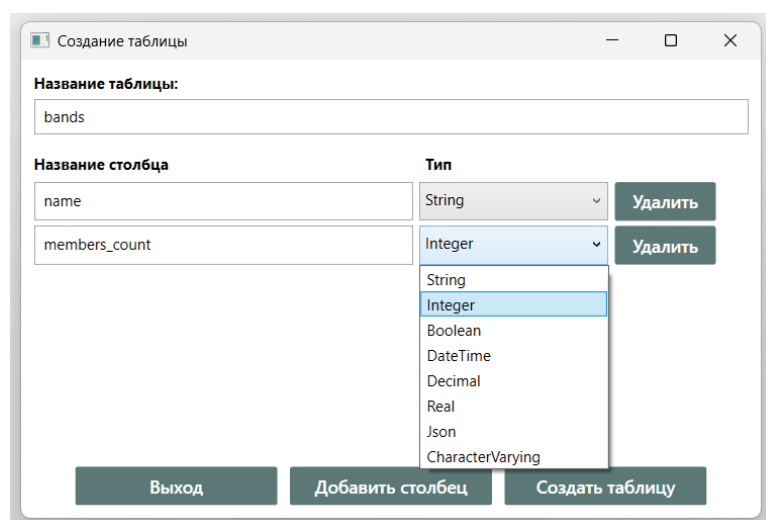


Рисунок 1.6 – Окно создания новой таблицы

Для удаления таблицы необходимо выбрать необходимую таблицу в списке и нажать на кнопку «Удалить таблицу» в области управления таблицами.

1.4 Добавление, редактирование и удаление строк

1.4.1 Добавление строк

Для добавления новой строки в таблицу необходимо выбрать таблицу в области управления таблицами, нажать на кнопку «Добавить строку». В таблице данных, отображенной в центральной части окна, появится новая пустая строка. Необходимо заполнить ячейки новой строки в соответствии с типом столбца (если в таблице есть столбец с именем id, его значение будет

сгенерировано автоматически). После заполнения всех ячеек необходимо нажать на кнопку «сохранить новую строку», после чего строка будет сохранена в базе данных.

1.4.2 Редактирование строк

Для редактирования существующей строки в таблице необходимо в списке таблиц выбрать нужную таблицу, дважды щелкнуть по ячейке, которую необходимо изменить, чтобы активировать режим редактирования, после чего ввести новое значение. После нажатия клавиши Enter или щелчка вне ячейки новое значение будет сохранено в базе данных. Если значение некорректно, появится сообщение об ошибке.

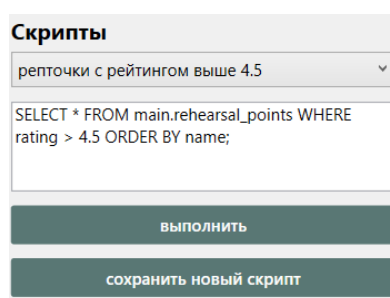
1.4.3 Удаление строк

Для удаления строки из таблицы необходимо в списке таблиц выбрать таблицу, из которой нужно удалить строку, выделить строку, которую необходимо удалить, щелкнув по ней левой кнопкой мыши, после чего нажать на кнопку «удалить строку».

1.5 Вывод результатов запросов

1.5.1 Вывод результатов существующих запросов

Для выполнения SQL-запроса в области управления скриптами в выпадающем списке необходимо выбрать соответствующее название запроса. SQL-запрос отобразится в области для скрипта, что отображено на рисунке 1.7. Далее необходимо нажать на кнопку «выполнить». Результат скрипта отобразится в центральной части окна.



Скрипты

репточки с рейтингом выше 4.5 ▾

```
SELECT * FROM main.rehearsal_points WHERE  
rating > 4.5 ORDER BY name;
```

выполнить

сохранить новый скрипт

Рисунок 1.7 – Отображение SQL-запроса

1.5.2 Сохранение новых запросов

Для сохранения новых SQL-запросов в выпадающем списке необходимо выбрать «новый скрипт» и ввести SQL-запрос в область скрипта, отображено

на рисунке 1.8. После этого необходимо нажать на кнопку «сохранить новый скрипт», появится окно для ввода имени скрипта, которое приведено на рисунке 1.9. Необходимо ввести название, которое будет отображаться в выпадающем списке, после чего нажать «ОК», скрипт будет внесен в список пользовательских и будет отображаться в общем выпадающем списке.

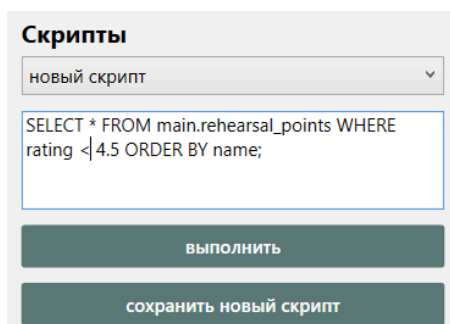


Рисунок 1.8 – Ввод нового скрипта

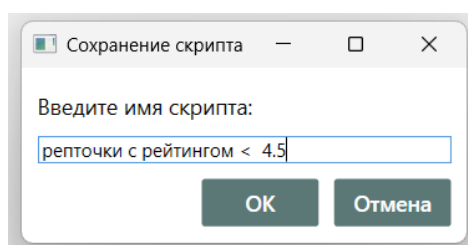


Рисунок 1.9 – Ввод имени скрипта

1.6 Создание резервной копии

Для создания резервной копии всей базы данных необходимо нажать на кнопку «Создать бэкап» в области резервного сохранения. После этого в случае успеха будет создан SQL-скрипт и появится уведомление об успешном создании бэкапа, отображенное на рисунке 1.10.

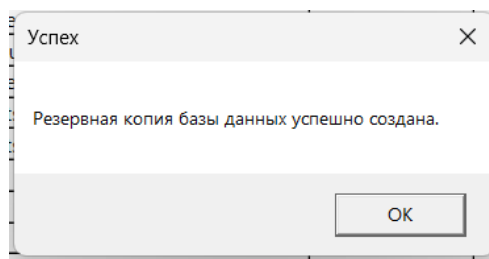


Рисунок 1.10 – Уведомление об успешном создании бэкапа

Также бэкапы создаются автоматически при удалении строк и столбцов, с целью предотвратить потерю ошибочно удаленных данных.

Для восстановления базы данных до состояния бэкапа, необходимо нажать на кнопку «восстановить бэкап», после чего будет отображено окно восстановления бэкапа, в котором отображены все сохраненные бэкапы, оно приведено на рисунке 1.11. Необходимо выбрать конкретный бэкап и нажать на кнопку «восстановить». Также бэкап можно удалить, нажав на кнопку окна восстановления «удалить».

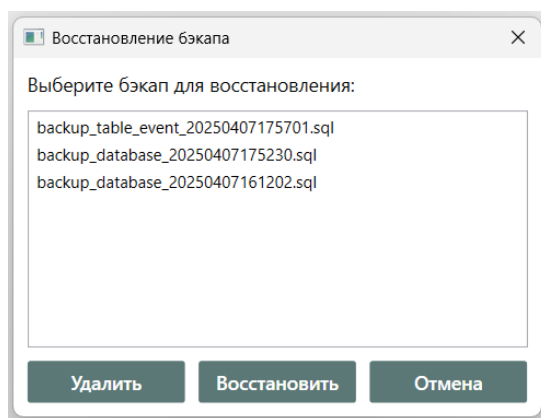


Рисунок 1.11 – Окно восстановления бэкапа

1.7 Механизм экспорта в файл Excel

Для экспорта данных таблицы в Excel необходимо выбрать таблицу для экспорта в списке таблиц, в области управления таблицами нажать на кнопку «Экспорт в Excel», после чего появится окно выбора, представленное на рисунке 1.12. В нем необходимо выбрать «Текущая таблица» для одной таблицы и «Вся база данных» для экспорта всех таблиц в один файл и нажать на кнопку «Экспорт». После чего в проводнике необходимо задать путь и имя файла. В случае экспорта результата скрипта, необходимо предварительно его выполнить, нажать на кнопку «Экспорт в Excel» и выбрать «Результат скрипта», после чего также в проводнике задать путь и имя файла.

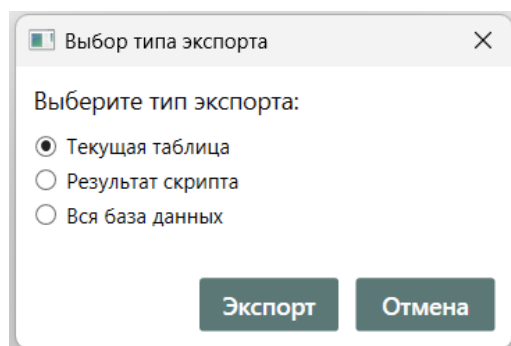


Рисунок 1.12 – Окно выбора типа экспорта

2 ЛИСТИНГ КОДА

Файл App.xaml:

```
001 <Application x:Class="RepBase.App"
002
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
003             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
004             xmlns:local="clr-namespace:RepBase"
005             StartupUri="MainWindow.xaml">
006     <Application.Resources>
007
008     </Application.Resources>
009 </Application>
```

Файл App.xaml.cs:

```
001 using System;
002 using System.Collections.Generic;
003 using System.Configuration;
004 using System.Data;
005 using System.Linq;
006 using System.Threading.Tasks;
007 using System.Windows;
008 namespace RepBase
009 {
010     /// <summary>
011     /// Логика взаимодействия для App.xaml
012     /// </summary>
013     public partial class App : Application
014     {
015     }
016 }
```

Файл Converters.cs:

```
001 using System;
002 using System.Globalization;
003 using System.Windows.Data;
004 namespace RepBase
005 {
006     public class NullToBooleanConverter : IValueConverter
007     {
008         public object Convert(object value, Type targetType, object
parameter, CultureInfo culture)
009         {
010             return value != null;
011         }
012         public object ConvertBack(object value, Type targetType,
object parameter, CultureInfo culture)
013         {
014             throw new NotImplementedException();
015         }
016     }
017 }
```

Файл CreateTableWindow.xaml:

```
001 <Window x:Class="RepBase.CreateTableWindow"
002
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
003             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
004             Title="Создание таблицы" Height="400" Width="600"
```

```

005         WindowStartupLocation="CenterOwner">
006     <Window.Resources>
007         <Style TargetType="Button">
008             <Setter Property="Background" Value="#486966"/>
009             <Setter Property="Foreground" Value="White"/>
010             <Setter Property="Padding" Value="12,6"/>
011             <Setter Property="Margin" Value="0"/>
012             <Setter Property="FontWeight" Value="SemiBold"/>
013             <Setter Property="BorderThickness" Value="0"/>
014             <Setter Property="Cursor" Value="Hand"/>
015             <Setter Property="FontSize" Value="14"/>
016             <Setter Property="Opacity" Value="0.9"/>
017             <Setter Property="Template">
018                 <Setter.Value>
019                     <ControlTemplate TargetType="Button">
020                         <Border Background="{TemplateBinding
Background}"
021                             BorderBrush="{TemplateBinding
BorderBrush}"
022                             BorderThickness="{TemplateBinding
BorderThickness}"
023                             Padding="12,6"
024                             CornerRadius="2">
025                             <ContentPresenter
HorizontalAlignment="Center"
026                             VerticalAlignment="Center"/>
027                         </Border>
028                     </ControlTemplate>
029                 </Setter.Value>
030             </Setter>
031             <Style.Triggers>
032                 <Trigger Property="IsMouseOver" Value="True">
033                     <Setter Property="Background" Value="#B0BEC5"/>
034                     <Setter Property="Opacity" Value="1"/>
035                 </Trigger>
036                 <Trigger Property="IsPressed" Value="True">
037                     <Setter Property="Background" Value="#889C9B"/>
038                 </Trigger>
039             </Style.Triggers>
040         </Style>
041     </Window.Resources>
042     <Grid Margin="10">
043         <Grid.RowDefinitions>
044             <RowDefinition Height="Auto"/>
045             <RowDefinition Height="*/>
046             <RowDefinition Height="Auto"/>
047         </Grid.RowDefinitions>
048         <StackPanel Grid.Row="0" Margin="0,0,0,10">
049             <TextBlock Text="Название таблицы:"
050                 FontWeight="Bold"/>
051             <TextBox Text="{Binding TableName,
UpdateSourceTrigger=PropertyChanged}" FontSize="12" Padding="5"
Margin="0,5"/>
052         </StackPanel>
053         <Grid Grid.Row="1">
054             <Grid.RowDefinitions>
055                 <RowDefinition Height="Auto"/>
056                 <RowDefinition Height="*/>
057             </Grid.RowDefinitions>
058             <StackPanel Grid.Row="0" Orientation="Horizontal"
Margin="0,0,0,5">

```

```

059             <TextBlock Text="Название столбца"
FontWeight="Bold" Width="310"/>
060             <TextBlock Text="Тип" FontWeight="Bold"
Width="150"/>
061         </StackPanel>
062         <ScrollViewer Grid.Row="1"
VerticalScrollBarVisibility="Auto">
063             <ItemsControl ItemsSource="{Binding Columns}">
064                 <ItemsControl.ItemTemplate>
065                     <DataTemplate>
066                         <StackPanel
Orientation="Horizontal" Margin="0,2">
067                             <TextBox Text="{Binding
ColumnName, UpdateSourceTrigger=PropertyChanged}"
068                                 Width="300" FontSize="12"
Padding="5" Margin="0,0,5,0"/>
069                             <ComboBox
ItemsSource="{Binding DataContext.ColumnTypes, RelativeSource={RelativeSource
AncestorType=Window}}"
070                                 SelectedItem="{Binding
ColumnType}"
071                                 Width="150" FontSize="12"
Padding="5" Margin="0,0,5,0"/>
072                             <Button
Content="Удалить"
073                                 Command="{Binding
DataContext.RemoveColumnCommand, RelativeSource={RelativeSource
AncestorType=Window}}"
074                                 CommandParameter="{Binding}"
075                                 Width="80"/>
076                         </StackPanel>
077                     </DataTemplate>
078                 </ItemsControl.ItemTemplate>
079             </ItemsControl>
080         </ScrollViewer>
081     </Grid>
082     <StackPanel Grid.Row="2" Orientation="Horizontal"
HorizontalAlignment="Center" Margin="0,10,0,0" Width="500" >
083         <Button Content="Выход"
084             Click="Cancel_Click"
085             Width="160" Margin="0,0,10,0"/>
086         <Button Content="Добавить столбец"
087             Command="{Binding AddColumnCommand}"
088             Width="160" Margin="0,0,10,0"/>
089         <Button Content="Создать таблицу"
090             Command="{Binding CreateTableCommand}"
091             CommandParameter="{Binding
RelativeSource={RelativeSource AncestorType=Window}}"
092             Width="160" Margin="0,0,10,0"/>
093     </StackPanel>
094 </Grid>
095 </Window>

```

Файл CreateTableWindow.xaml.cs:

```

001 using RepBase.Data;
002 using RepBase.ViewModels;
003 using System.Windows;
004 namespace RepBase
005 {
006     public partial class CreateTableWindow : Window
007     {

```

```

008         public CreateTableWindow(DatabaseManager databaseManager)
009         {
010             InitializeComponent();
011             DataContext = new CreateTableViewModel(databaseManager);
012         }
013         private void Cancel_Click(object sender, RoutedEventArgs e)
014         {
015             Close();
016         }
017         private void Button_Click(object sender, RoutedEventArgs e)
018         {
019         }
020     }
021 }

```

Файл MainWindow.xaml:

```

001 <Window x:Class="RepBase.MainWindow"
002
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
003     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
004     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
005     xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
006     xmlns:local="clr-namespace:RepBase"
007     mc:Ignorable="d"
008     Title="RepBase"
009     WindowState="Maximized"
010 >
011     <Window.Resources>
012         <ResourceDictionary>
013             <ResourceDictionary.MergedDictionaries>
014                 <ResourceDictionary
Source="/Resources/Styles.xaml"/>
015             </ResourceDictionary.MergedDictionaries>
016         </ResourceDictionary>
017     </Window.Resources>
018     <Grid>
019         <Grid.ColumnDefinitions>
020             <ColumnDefinition Width="1*" />
021             <ColumnDefinition Width="4*" />
022         </Grid.ColumnDefinitions>
023         <StackPanel Grid.Column="0" Margin="5" Background="#f0f0f0">
024             <StackPanel Margin="5">
025                 <TextBlock Text="Таблицы" FontWeight="Bold"
FontSize="16" Margin="5,0"/>
026                 <ScrollViewer Height="150" Margin="5">
027                     <ItemsControl ItemsSource="{Binding
TableItems}">
028                         <ItemsControl.ItemTemplate>
029                             <DataTemplate>
030                                 <Button Content="{Binding
TableName}"
031                                     Command="{Binding
DataContext.SelectCommand, RelativeSource={RelativeSource
AncestorType=Window}}">
032                                     CommandParameter="{Binding}"
033                                     Margin="5"
034                                     Background="#889C9B"/>
035                             </DataTemplate>
036                         </ItemsControl.ItemTemplate>
037                     </ItemsControl>
038                 </ScrollViewer>

```

```

039             <Button Content="создать новую таблицу"
040                     Command="{Binding CreateTableCommand}"
041                     Margin="5" />
042             <Button Content="удалить таблицу"
043                     Command="{Binding DeleteTableCommand}"
044                     Margin="5" />
045         </StackPanel>
046
047         <StackPanel Margin="5">
048             <TextBlock Text="Содержание таблицы"
FontWeight="Bold" FontSize="16" Margin="5,0" />
049             <Button Content="добавить строку"
050                     Command="{Binding AddRowCommand}"
051                     Margin="5" />
052             <Button Content="сохранить новую строку"
053                     Command="{Binding SaveNewRowCommand}"
054                     CommandParameter="{Binding
ElementName=dataGrid, Path=SelectedItem}"
055                     Margin="5" />
056             <Button Content="удалить строку"
057                     Command="{Binding DeleteRowCommand}"
058                     CommandParameter="{Binding
ElementName=dataGrid, Path=SelectedItem}"
059                     Margin="5" />
060         </StackPanel>
061         <StackPanel Margin="5">
062             <TextBlock Text="Скрипты" FontWeight="Bold"
FontSize="16" Margin="5,0"/>
063             <ComboBox x:Name="scriptsComboBox"
064                     Margin="5"
065                     FontSize="12"
066                     ItemsSource="{Binding ScriptNames,
Mode=OneWay}"
067                     SelectedItem="{Binding SelectedScriptName,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
068             SelectionChanged="ScriptsComboBox_SelectionChanged"/>
069             <TextBox x:Name="scriptTextBox"
070                     Margin="5"
071                     Height="65"
072                     AcceptsReturn="True"
073                     AcceptsTab="True"
074                     TextWrapping="Wrap"
075                     IsReadOnly="False"
076                     Text="{Binding CurrentScript, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
077                     FontSize="12"
078                     TextChanged="ScriptTextBox_TextChanged"/>
079             <Button Content="выполнить"
080                     Command="{Binding ExecuteScriptCommand}"
081                     Margin="5"/>
082             <Button Content="сохранить новый скрипт"
083                     Command="{Binding SaveScriptCommand}"
084                     Margin="5"/>
085         </StackPanel>
086         <StackPanel Margin="5">
087             <TextBlock Text="Резервное сохранение"
FontWeight="Bold" FontSize="16" Margin="5,0"/>
088             <Button Content="создать бэкап" Command="{Binding
CreateBackupCommand}" Margin="5"/>
089             <Button Content="восстановить бэкап"
Command="{Binding ShowRestoreBackupCommand}" Margin="5"/>

```

```

090             <Button Content="экспорт в Excel" Command="{Binding
ShowExportOptionsCommand}" Margin="5"/>
091         </StackPanel>
092     </StackPanel>
093     <DataGrid x:Name="dataGrid"
094         Grid.Column="1"
095         Margin="5"
096         ItemsSource="{Binding TableData, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
097         AutoGenerateColumns="True"
098
AutoGeneratingColumn="DataGrid_AutoGeneratingColumn"
099         HeadersVisibility="Column"
100         RowHeaderWidth="0"
101         HorizontalScrollBarVisibility="Auto"
102         VerticalScrollBarVisibility="Auto"
103         CanUserAddRows="False"
104         CanUserDeleteRows="False"
105         IsReadOnly="False"
106         CellEditEnding="DataGrid_CellEditEnding"
107         SelectionMode="Single"
108         SelectionUnit="FullRow"/>
109     </Grid>
110 </Window>

```

Файл MainWindow.xaml.cs:

```

001 using RepBase.ViewModels;
002 using System;
003 using System.Data;
004 using System.Windows;
005 using System.Windows.Controls;
006 namespace RepBase
007 {
008     public partial class MainWindow : Window
009     {
010         private bool _isUpdatingScriptText = false;
011         public MainWindow()
012         {
013             InitializeComponent();
014             DataContext = new MainViewModel();
015             var viewModel = DataContext as MainViewModel;
016             if (viewModel != null)
017             {
018                 Console.WriteLine($"ScriptNames count after
initialization: {viewModel.ScriptNames.Count}");
019                 foreach (var script in viewModel.ScriptNames)
020                 {
021                     Console.WriteLine($"Script: {script}");
022                 }
023             }
024         }
025         private void DataGrid_CellEditEnding(object sender,
DataGridCellEditEndingEventArgs e)
026         {
027             if (e.EditAction == DataGridEditAction.Commit)
028             {
029                 var viewModel = DataContext as MainViewModel;
030                 var row = e.Row.Item as DataRowView;
031                 var column = e.Column as DataGridColumn;
032                 if (viewModel != null && row != null && column !=
null)
033             {

```

```

034          // Проверяем, является ли строка "новой"
035          bool isNewRow = true;
036          foreach (DataColumn col in
row.Row.Table.Columns)
037              {
038                  if (col.ColumnName != "id" &&
row.Row[col.ColumnName] != DBNull.Value)
039                      {
040                          isNewRow = false;
041                          break;
042                      }
043              }
044          if (isNewRow)
045              {
046              // Если строка новая, не вызываем UpdateCell
047              return;
048              }
049          var columnName = column.Header.ToString();
050          object newValue = null;
051          if (e.Column is DataGridCheckBoxColumn)
052              {
053                  newValue = (e.EditingElement as
CheckBox)?.IsChecked;
054              }
055          else
056              {
057                  newValue = (e.EditingElement as
TextBox)?.Text;
058              }
059          var oldValue = row[columnName];
060          if (!Equals(newValue, oldValue))
061              {
062                  var args = new CellUpdateArgs
063                  {
064                      Row = row.Row,
065                      ColumnName = columnName,
066                      NewValue = newValue
067                  };
068                  viewModel.UpdateCellCommand.Execute(args);
069              }
070          }
071      }
072  }
073  private void DataGrid_AutoGeneratingColumn(object sender,
DataGridAutoGeneratingColumnEventArgs e)
074      {
075          e.Column.Header = e.PropertyName;
076      }
077  private void ScriptsComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
078      {
079          var viewModel = DataContext as MainViewModel;
080          if (viewModel != null && scriptsComboBox.SelectedItem !=
null)
081              {
082                  _isUpdatingScriptText = true;
083                  viewModel.UpdateCurrentScript();
084                  _isUpdatingScriptText = false;
085              }
086      }
087  private void ScriptTextBox_TextChanged(object sender,
TextChangedEventArgs e)
088      {

```



```

089             if (_isUpdatingScriptText) return;
090             var viewModel = DataContext as MainViewModel;
091             if (viewModel != null)
092             {
093                 if (scriptTextBox.Text != viewModel.CurrentScript &&
viewModel.SelectedScriptName != "новый скрипт")
094                 {
095                     viewModel.SelectedScriptName = "новый скрипт";
096                 }
097             }
098         }
099     }
100 }

```

Файл RepBase.csproj:

```

001 <?xml version="1.0" encoding="utf-8"?>
002 <Project ToolsVersion="15.0"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
003     <Import
Project="$ (MSBuildExtensionsPath) \ $ (MSBuildToolsVersion) \ Microsoft.Common.pro
ps"
Condition="Exists (' $ (MSBuildExtensionsPath) \ $ (MSBuildToolsVersion) \ Microsoft.
Common.props') " />
004     <PropertyGroup>
005         <Configuration Condition=" '$ (Configuration)' == ' '
">Debug</Configuration>
006         <Platform Condition=" '$ (Platform)' == ' ' ">AnyCPU</Platform>
007         <ProjectGuid>{D2722421-0F04-40F9-B2C2-
E137E5BD83D9}</ProjectGuid>
008         <OutputType>WinExe</OutputType>
009         <RootNamespace>RepBase</RootNamespace>
010         <AssemblyName>RepBase</AssemblyName>
011         <TargetFrameworkVersion>v4.7.2</TargetFrameworkVersion>
012         <FileAlignment>512</FileAlignment>
013         <ProjectTypeGuids>{60dc8134-eba5-43b8-bcc9-
bb4bc16c2548};{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}</ProjectTypeGuids>
014         <WarningLevel>4</WarningLevel>
015
<AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
016         <Deterministic>true</Deterministic>
017     </PropertyGroup>
018     <PropertyGroup Condition=" '$ (Configuration) | $ (Platform)' ==
' Debug | AnyCPU' ">
019         <PlatformTarget>AnyCPU</PlatformTarget>
020         <DebugSymbols>true</DebugSymbols>
021         <DebugType>full</DebugType>
022         <Optimize>false</Optimize>
023         <OutputPath>bin\Debug\</OutputPath>
024         <DefineConstants>DEBUG;TRACE</DefineConstants>
025         <ErrorReport>prompt</ErrorReport>
026         <WarningLevel>4</WarningLevel>
027     </PropertyGroup>
028     <PropertyGroup Condition=" '$ (Configuration) | $ (Platform)' ==
' Release | AnyCPU' ">
029         <PlatformTarget>AnyCPU</PlatformTarget>
030         <DebugType>pdbonly</DebugType>
031         <Optimize>true</Optimize>
032         <OutputPath>bin\Release\</OutputPath>
033         <DefineConstants>TRACE</DefineConstants>
034         <ErrorReport>prompt</ErrorReport>
035         <WarningLevel>4</WarningLevel>
036     </PropertyGroup>

```

```

037 <PropertyGroup>
038   <EnableEPPlusLicense>true</EnableEPPlusLicense>
039 </PropertyGroup>
040 <ItemGroup>
041   <Reference Include="System" />
042   <Reference Include="System.Data" />
043   <Reference Include="System.Xml" />
044   <Reference Include="Microsoft.CSharp" />
045   <Reference Include="System.Core" />
046   <Reference Include="System.Xml.Linq" />
047   <Reference Include="System.Data.DataSetExtensions" />
048   <Reference Include="System.Net.Http" />
049   <Reference Include="System.Xaml">
050     <RequiredTargetFramework>4.0</RequiredTargetFramework>
051   </Reference>
052   <Reference Include="WindowsBase" />
053   <Reference Include="PresentationCore" />
054   <Reference Include="PresentationFramework" />
055 </ItemGroup>
056 <ItemGroup>
057   <ApplicationDefinition Include="App.xaml">
058     <Generator>MSBuild:Compile</Generator>
059     <SubType>Designer</SubType>
060   </ApplicationDefinition>
061   <Compile Include="Converters.cs" />
062   <Compile Include="CreateTableWindow.xaml.cs">
063     <DependentUpon>CreateTableWindow.xaml</DependentUpon>
064   </Compile>
065   <Compile Include="ExportOptionsWindow.xaml.cs">
066     <DependentUpon>ExportOptionsWindow.xaml</DependentUpon>
067   </Compile>
068   <Compile Include="RestoreBackupWindow.xaml.cs">
069     <DependentUpon>RestoreBackupWindow.xaml</DependentUpon>
070   </Compile>
071   <Compile Include="ScriptNameDialog.xaml.cs">
072     <DependentUpon>ScriptNameDialog.xaml</DependentUpon>
073   </Compile>
074   <Compile Include="Services\BackupService.cs" />
075   <Compile Include="Services\ExportService.cs" />
076   <Compile Include="Services\ScriptService.cs" />
077   <Compile Include="Services\TableService.cs" />
078   <Compile Include="ViewModels\CreateTableViewModel.cs" />
079   <Compile Include="ViewModels\MainViewModel.cs" />
080   <Compile Include="ViewModels\RestoreBackupViewModel.cs" />
081   <Page Include="CreateTableWindow.xaml">
082     <Generator>XamlIntelliSenseFileGenerator</Generator>
083     <SubType>Designer</SubType>
084   </Page>
085   <Page Include="ExportOptionsWindow.xaml">
086     <Generator>MSBuild:Compile</Generator>
087   </Page>
088   <Page Include="MainWindow.xaml">
089     <Generator>MSBuild:Compile</Generator>
090     <SubType>Designer</SubType>
091   </Page>
092   <Compile Include="App.xaml.cs">
093     <DependentUpon>App.xaml</DependentUpon>
094     <SubType>Code</SubType>
095   </Compile>
096   <Compile Include="Data\DatabaseManager.cs" />
097   <Compile Include="MainWindow.xaml.cs">
098     <DependentUpon>MainWindow.xaml</DependentUpon>
099     <SubType>Code</SubType>

```

```

100     </Compile>
101     <Page Include="Resources\Styles.xaml">
102         <Generator>MSBuild:Compile</Generator>
103     </Page>
104     <Page Include="RestoreBackupWindow.xaml">
105         <Generator>MSBuild:Compile</Generator>
106     </Page>
107     <Page Include="ScriptNameDialog.xaml">
108         <Generator>MSBuild:Compile</Generator>
109     </Page>
110 </ItemGroup>
111 <ItemGroup>
112     <Compile Include="Models\ColumnModel.cs" />
113     <Compile Include="Models\RowModel.cs" />
114     <Compile Include="Models\TableModel.cs" />
115     <Compile Include="Properties\AssemblyInfo.cs">
116         <SubType>Code</SubType>
117     </Compile>
118     <Compile Include="Properties\Resources.Designer.cs">
119         <AutoGen>True</AutoGen>
120         <DesignTime>True</DesignTime>
121         <DependentUpon>Resources.resx</DependentUpon>
122     </Compile>
123     <Compile Include="Properties\Settings.Designer.cs">
124         <AutoGen>True</AutoGen>
125         <DependentUpon>Settings.settings</DependentUpon>
126         <DesignTimeSharedInput>True</DesignTimeSharedInput>
127     </Compile>
128     <EmbeddedResource Include="Properties\Resources.resx">
129         <Generator>ResXFileCodeGenerator</Generator>
130         <LastGenOutput>Resources.Designer.cs</LastGenOutput>
131     </EmbeddedResource>
132     <None Include="default_scripts.json">
133         <CopyToOutputDirectory>Always</CopyToOutputDirectory>
134     </None>
135     <None Include="Properties\Settings.settings">
136         <Generator>SettingsSingleFileGenerator</Generator>
137         <LastGenOutput>Settings.Designer.cs</LastGenOutput>
138     </None>
139 </ItemGroup>
140 <ItemGroup>
141     <None Include="App.config" />
142 </ItemGroup>
143 <ItemGroup>
144     <Folder Include="Views\" />
145 </ItemGroup>
146 <ItemGroup>
147     <PackageReference Include="EPPlus">
148         <Version>7.2.2</Version>
149     </PackageReference>
150     <PackageReference Include="Npgsql" Version="8.0.7" />
151 </ItemGroup>
152 <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
153 </Project>

```

Файл RestoreBackupWindow.xaml:

```

001 <Window x:Class="RepBase.RestoreBackupWindow"
002     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
003         xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
004         xmlns:local="clr-namespace:RepBase"
005         Title="Восстановление бэкапа"

```

```

006         Height="300"
007         Width="400"
008         WindowStartupLocation="CenterOwner"
009         ResizeMode="NoResize">
010     <Window.Resources>
011         <local:NullToBooleanConverter
x:Key="NullToBooleanConverter"/>
012         <Style TargetType="Button">
013             <Setter Property="Background" Value="#486966"/>
014             <Setter Property="Foreground" Value="White"/>
015             <Setter Property="Padding" Value="12,6"/>
016             <Setter Property="Margin" Value="0"/>
017             <Setter Property="FontWeight" Value="SemiBold"/>
018             <Setter Property="BorderThickness" Value="0"/>
019             <Setter Property="Cursor" Value="Hand"/>
020             <Setter Property="FontSize" Value="14"/>
021             <Setter Property="Opacity" Value="0.9"/>
022             <Setter Property="Template">
023                 <Setter.Value>
024                     <ControlTemplate TargetType="Button">
025                         <Border Background="{TemplateBinding
Background}"
026                             BorderBrush="{TemplateBinding
BorderBrush}"
027                             BorderThickness="{TemplateBinding
BorderThickness}"
028                             Padding="12,6"
029                             CornerRadius="2">
030                             <ContentPresenter
HorizontalAlignment="Center"
031                             VerticalAlignment="Center"/>
032                         </Border>
033                     </ControlTemplate>
034                 </Setter.Value>
035             </Setter>
036             <Style.Triggers>
037                 <Trigger Property="IsMouseOver" Value="True">
038                     <Setter Property="Background" Value="#B0BEC5"/>
039                     <Setter Property="Opacity" Value="1"/>
040                 </Trigger>
041                 <Trigger Property="IsPressed" Value="True">
042                     <Setter Property="Background" Value="#889C9B"/>
043                 </Trigger>
044             </Style.Triggers>
045         </Style>
046     </Window.Resources>
047     <Grid Margin="10">
048         <Grid.RowDefinitions>
049             <RowDefinition Height="Auto"/>
050             <RowDefinition Height="*/>
051             <RowDefinition Height="Auto"/>
052         </Grid.RowDefinitions>
053         <TextBlock Grid.Row="0"
054             Text="Выберите бэкап для восстановления:"
055             FontSize="14"
056             Margin="0,0,0,10"/>
057         <ListBox Grid.Row="1"
058             x:Name="backupListBox"
059             ItemsSource="{Binding Backups}"
060             DisplayMemberPath="Name"
061             SelectedItem="{Binding SelectedBackup,
Mode=TwoWay}"

```

```

062             Margin="0,0,0,10"/>
063         <StackPanel Grid.Row="2"
064             Orientation="Horizontal"
065             HorizontalAlignment="Center">
066             <Button Content="Удалить"
067                 Width="115"
068                 Margin="0,0,10,0"
069                 Command="{Binding DeleteBackupCommand}"
070                 IsEnabled="{Binding SelectedBackup,
Converter={StaticResource NullToBooleanConverter}}"/>
071             <Button Content="Восстановить"
072                 Width="115"
073                 Margin="0,0,10,0"
074                 Command="{Binding RestoreBackupCommand}"
075                 IsEnabled="{Binding SelectedBackup,
Converter={StaticResource NullToBooleanConverter}}"/>
076             <Button Content="Отмена"
077                 Width="115"
078                 Click="Cancel_Click"/>
079         </StackPanel>
080     </Grid>
081 </Window>

```

Файл RestoreBackupWindow.xaml.cs:

```

001 using System.Windows;
002 namespace RepBase
003 {
004     public partial class RestoreBackupWindow : Window
005     {
006         public RestoreBackupWindow(RestoreBackupViewModel viewModel)
007         {
008             InitializeComponent();
009             DataContext = viewModel;
010         }
011         private void Cancel_Click(object sender, RoutedEventArgs e)
012         {
013             Close();
014         }
015     }
016 }

```

Файл ScriptNameDialog.xaml:

```

001 <Window x:Class="RepBase.ScriptNameDialog"
002
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
003     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
004     Title="Сохранение скрипта"
005     Width="300"
006     Height="150"
007     WindowStartupLocation="CenterOwner">
008     <Window.Resources>
009         <!-- Стилль для кнопок -->
010         <Style x:Key="DialogButtonStyle" TargetType="Button">
011             <Setter Property="Background" Value="#486966"/>
012             <Setter Property="Foreground" Value="White"/>
013             <Setter Property="Padding" Value="12,6"/>
014             <Setter Property="Margin" Value="0"/>
015             <Setter Property="FontWeight" Value="SemiBold"/>
016             <Setter Property="BorderThickness" Value="0"/>
017             <Setter Property="Cursor" Value="Hand"/>
018             <Setter Property="FontSize" Value="14"/>

```

```

019         <Setter Property="Opacity" Value="0.9"/>
020         <Setter Property="Width" Value="75"/>
021         <Setter Property="Template">
022             <Setter.Value>
023                 <ControlTemplate TargetType="Button">
024                     <Border Background="{TemplateBinding
Background}"
025                         BorderBrush="{TemplateBinding
BorderBrush}"
026                         BorderThickness="{TemplateBinding
BorderThickness}"
027                         Padding="12,6"
028                         CornerRadius="2">
029                         <ContentPresenter
HorizontalAlignment="Center"
030                         VerticalAlignment="Center"/>
031                     </Border>
032                 </ControlTemplate>
033             </Setter.Value>
034         </Setter>
035         <Style.Triggers>
036             <Trigger Property="IsMouseOver" Value="True">
037                 <Setter Property="Background" Value="#B0BEC5"/>
038                 <Setter Property="Opacity" Value="1"/>
039             </Trigger>
040             <Trigger Property="IsPressed" Value="True">
041                 <Setter Property="Background" Value="#889C9B"/>
042             </Trigger>
043         </Style.Triggers>
044     </Style>
045 </Window.Resources>
046 <StackPanel Margin="10">
047     <TextBlock Text="Введите имя скрипта:" Margin="0, 5"
FontSize="14"/>
048     <TextBox x:Name="ScriptNameTextBox" Margin="0,5,0,10"/>
049     <StackPanel Orientation="Horizontal"
HorizontalAlignment="Right">
050         <Button Content="OK"
051             Style="{StaticResource DialogButtonStyle}"
052             Margin="0,0,10,0"
053             Click="OkButton_Click"/>
054         <Button Content="Отмена"
055             Style="{StaticResource DialogButtonStyle}"
056             Click="CancelButton_Click"/>
057     </StackPanel>
058 </StackPanel>
059 </Window>

```

Файл ScriptNameDialog.xaml.cs:

```

001 using System.Windows;
002 namespace RepBase
003 {
004     public partial class ScriptNameDialog : Window
005     {
006         public string ScriptName => ScriptNameTextBox.Text;
007         public ScriptNameDialog()
008         {
009             InitializeComponent();
010             Owner = Application.Current.MainWindow;
011         }

```

```

012         private void OkButton_Click(object sender, RoutedEventArgs
e)
013         {
014             DialogResult = true;
015             Close();
016         }
017         private void CancelButton_Click(object sender,
RoutedEventArgs e)
018         {
019             DialogResult = false;
020             Close();
021         }
022     }
023 }

```

Файл DatabaseManager.cs:

```

001 using System;
002 using System.Collections.Generic;
003 using System.Data;
004 using System.Linq;
005 using System.Text;
006 using System.Threading.Tasks;
007 using System.Windows.Controls;
008 using Npgsql;
009 using RepBase.Models;
010 namespace RepBase.Data
011 {
012     public class DatabaseManager
013     {
014         private readonly string _connectionString;
015         public DatabaseManager(string connectionString)
016         {
017             _connectionString = connectionString;
018         }
019         public NpgsqlConnection GetConnection()
020         {
021             return new NpgsqlConnection(_connectionString);
022         }
023         public void ExecuteNonQuery(string query)
024         {
025             try
026             {
027                 using (var connection = new
NpgsqlConnection(_connectionString))
028                 {
029                     connection.Open();
030                     using (var command = new NpgsqlCommand(query,
connection))
031                     {
032                         command.ExecuteNonQuery();
033                     }
034                 }
035                 Console.WriteLine($"Query complited: {query}");
036             }
037             catch (NpgsqlException ex)
038             {
039                 Console.WriteLine($"Error executing npsql query:
{ex.Message}");
040             }
041             catch (Exception ex)
042             {

```

```

043             Console.WriteLine($"Error executing query:
{ex.Message}");
044         }
045     }
046     public DataTable ExecuteQuery(string query)
047     {
048         try
049         {
050             using (var connection = new
NpgsqlConnection(_connectionString))
051             {
052                 connection.Open();
053                 using (var command = new NpgsqlCommand(query,
connection))
054                 {
055                     using (var reader = command.ExecuteReader())
056                     {
057                         var dataTable = new DataTable();
058                         dataTable.Load(reader);
059                         Console.WriteLine($"Query complited:
{query}");
060                         return dataTable;
061                     }
062                 }
063             }
064         }
065         catch (NpgsqlException ex)
066         {
067             Console.WriteLine($"Error executing npsql query:
{ex.Message}");
068         }
069         catch (Exception ex)
070         {
071             Console.WriteLine($"Error executing query:
{ex.Message}");
072         }
073         return new DataTable();
074     }
075     public void ExecuteNonQueryWithParams(string query,
List<Npgsql.NpgsqlParameter> parameters)
076     {
077         try
078         {
079             using (var connection = new
NpgsqlConnection(_connectionString))
080             {
081                 connection.Open();
082                 using (var command = new NpgsqlCommand(query,
connection))
083                 {
084                     command.Parameters.AddRange(parameters.ToArray());
085                     command.ExecuteNonQuery();
086                 }
087             }
088         }
089         catch (Exception ex)
090         {
091             Console.WriteLine($"Error executing query:
{ex.Message}");
092             throw;
093         }
094     }

```



```

095         public object ExecuteScalar(string query,
List<Npgsql.NpgsqlParameter> parameters)
096     {
097         try
098         {
099             using (var connection = new
NpgsqlConnection(_connectionString))
100             {
101                 connection.Open();
102                 using (var command = new NpgsqlCommand(query,
connection))
103                 {
104                     command.Parameters.AddRange(parameters.ToArray());
105                     return command.ExecuteScalar(); //
Возвращаем одно значение
106                 }
107             }
108         }
109         catch (Exception ex)
110         {
111             Console.WriteLine($"Error executing query:
{ex.Message}");
112             throw;
113         }
114     }
115     public List<TableModel> LoadTables()
116     {
117         var tables = new List<TableModel>();
118         LoadTableNames(tables);
119         LoadTableColumns(tables);
120         foreach (var tableModel in tables)
121         {
122             LoadTableData(tableModel);
123         }
124         return tables;
125     }
126     public void LoadTableNames(List<TableModel> tables)
127     {
128         try
129         {
130             string query = "SELECT table_name FROM
information_schema.tables WHERE table_schema = 'main'";
131             using (var connection = new
NpgsqlConnection(_connectionString))
132             {
133                 connection.Open();
134                 using (var command = new NpgsqlCommand(query,
connection))
135                 {
136                     using (var reader = command.ExecuteReader())
137                     {
138                         while (reader.Read())
139                         {
140                             var tableName = reader.GetString(0);
141                             tables.Add(new TableModel { TableName =
tableName, Columns = new List<ColumnModel>(), Rows = new List<RowModel>() });
142                         }
143                     }
144                     connection.Close();
145                 }
146             }
147         }
148         catch (NpgsqlException ex)
149         {

```

```

148             Console.WriteLine($"Error executing query:
{ex.Message}");
149         }
150     }
151     public void LoadTableColumns(List<TableModel> tables)
152     {
153         try
154         {
155             using (var connection = new
NpgsqlConnection(_connectionString))
156             {
157                 connection.Open();
158                 foreach (var tableModel in tables)
159                 {
160                     Console.WriteLine($"Loading
{tableModel.TableName} columns:");
161                     string queryColumns = $"SELECT column_name,
data_type FROM information_schema.columns WHERE table_name =
'{tableModel.TableName}' AND table_schema = 'main'";
162                     using (var commandColumns = new
NpgsqlCommand(queryColumns, connection))
163                     using (var readerColumns =
commandColumns.ExecuteReader())
164                     {
165                         while (readerColumns.Read())
166                         {
167                             var columnName =
readerColumns.GetString(0);
168                             var dataType =
readerColumns.GetString(1);
169                             var columnType =
MapDataTypeToColumnType(dataType);
170                             tableModel.Columns.Add(new
ColumnModel(columnName, columnType));
171                             Console.WriteLine($"Added column
{columnName} with type {columnType}");
172                         }
173                     }
174                 }
175             }
176         }
177         catch (NpgsqlException ex)
178         {
179             Console.WriteLine($"Error executing query:
{ex.Message}");
180             throw;
181         }
182     }
183 }
184 private ColumnType MapDataTypeToColumnType(string dataType)
185 {
186     dataType = dataType.ToLower();
187     switch (dataType)
188     {
189         case "character varying":
190             return ColumnType.CharacterVarying;
191         case "text":
192             return ColumnType.String;
193         case "integer":
194             return ColumnType.Integer;
195         case "real":
196             return ColumnType.Real;
197         case "boolean":

```

```

198             return ColumnType.Boolean;
199         case "timestamp without time zone":
200             return ColumnType.DateTime;
201         case "numeric":
202             return ColumnType.Decimal;
203         case "json":
204             return ColumnType.Json;
205         default:
206             throw new ArgumentException($"Unsupported data
type: {dataType}");
207     }
208 }
209 }
210 public void LoadTableData(TableModel tableModel)
211 {
212     try
213     {
214         using (var connection = new
NpgsqlConnection(_connectionString))
215         {
216             connection.Open();
217             var tableName = tableModel.TableName;
218             string queryRows = $"SELECT * FROM
main.{tableName}";
219             using (var commandRows = new
NpgsqlCommand(queryRows, connection))
220             using (var readerRows =
commandRows.ExecuteReader())
221             {
222                 while (readerRows.Read())
223                 {
224                     var row = new RowModel();
225                     for (int i = 0; i <
readerRows.FieldCount; i++)
226                     {
227                         row.Values[readerRows.GetName(i)] =
readerRows.GetValue(i);
228                     }
229                     tableModel.Rows.Add(row);
230                 }
231             }
232         }
233     }
234     catch (NpgsqlException ex)
235     {
236         Console.WriteLine($"Error executing query:
{ex.Message}");
237     }
238     Console.WriteLine($"Loaded {tableModel.Rows.Count} rows
from {tableModel.TableName}.");
239 }
240 public DataTable GetTableData(string tableName)
241 {
242     string query = $"SELECT * FROM main.{tableName}";
243     return ExecuteQuery(query);
244 }
245 public void CreateTable(string tableName, string
tableDefinition)
246 {
247     string query = $"CREATE TABLE IF NOT EXISTS
main.{tableName} ({tableDefinition});";
248     ExecuteNonQuery(query);
249 }

```

```

250         public void DropTable(string tableName)
251         {
252             try
253             {
254                 using (var connection = new
NpgsqlConnection(_connectionString))
255                 {
256                     connection.Open();
257                     string query = $"DROP TABLE IF EXISTS
main.{tableName} CASCADE;";
258                     Console.WriteLine($"Executing query: {query}");
// Для отладки
259                     using (var command = new NpgsqlCommand(query,
connection))
260                     {
261                         int rowsAffected =
command.ExecuteNonQuery();
262                         Console.WriteLine($"DropTable:
{rowsAffected} rows affected"); // Для отладки
263                     }
264                 }
265             }
266             catch (Exception ex)
267             {
268                 Console.WriteLine($"Error in DropTable:
{ex.Message}"); // Для отладки
269                 throw; // Перебрасываем исключение, чтобы увидеть
его в MainViewModel
270             }
271         }
272         public int GetNextId(string tableName)
273         {
274             var dataTable = GetTableData(tableName);
275             if
(!dataTable.Columns.Contains("id") || dataTable.Rows.Count == 0)
276             {
277                 return 1;
278             }
279             return dataTable.AsEnumerable().Max(row =>
row.Field<int>("id")) + 1;
280         }
281     }
282 }

```

Файл ColumnModel.cs:

```

001 using System;
002 using System.Collections.Generic;
003 using System.ComponentModel;
004 using System.Linq;
005 using System.Runtime.CompilerServices;
006 using System.Text;
007 using System.Threading.Tasks;
008 using RepBase.Services;
009 namespace RepBase.Models
010 {
011     public enum ColumnType
012     {
013         String,
014         Integer,
015         Boolean,
016         DateTime,
017         Decimal,

```

```

018         Real,
019         Json,
020         CharacterVarying,
021     }
022     public class ColumnModel : INotifyPropertyChanged
023     {
024         private string _columnName;
025         private ColumnType _columnType;
026         public string ColumnName
027         {
028             get => _columnName;
029             set { _columnName = value; OnPropertyChanged(); }
030         }
031         public ColumnType ColumnType
032         {
033             get => _columnType;
034             set { _columnType = value; OnPropertyChanged(); }
035         }
036         public ColumnModel(string columnName, ColumnType columnType)
037         {
038             ColumnName = columnName;
039             ColumnType = columnType;
040         }
041         public event PropertyChangedEventHandler PropertyChanged;
042         protected virtual void OnPropertyChanged([CallerMemberName]
string propertyName = null)
043         {
044             PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
045         }
046     }
047 }

```

Файл RowModel.cs:

```

001 using System;
002 using System.Collections.Generic;
003 using System.ComponentModel;
004 using System.Linq;
005 using System.Runtime.CompilerServices;
006 using System.Text;
007 using System.Threading.Tasks;
008 namespace RepBase.Models
009 {
010     public class RowModel : INotifyPropertyChanged
011     {
012         private Dictionary<string, object> _values;
013         public Dictionary<string, object> Values
014         {
015             get => _values;
016             set { _values = value; OnPropertyChanged(); }
017         }
018         public RowModel()
019         {
020             Values = new Dictionary<string, object>();
021         }
022         public event PropertyChangedEventHandler PropertyChanged;
023         protected virtual void OnPropertyChanged([CallerMemberName]
string propertyName = null)
024         {
025             PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
026         }
027     }
028 }

```

```
027     }
028 }
```

Файл TableModel.cs:

```
001 using System;
002 using System.Collections.Generic;
003 using System.ComponentModel;
004 using System.Linq;
005 using System.Runtime.CompilerServices;
006 using System.Text;
007 using System.Threading.Tasks;
008 namespace RepBase.Models
009 {
010     public class TableModel : INotifyPropertyChanged
011     {
012         private string _tableName;
013         private List<ColumnModel> _columns;
014         private List<RowModel> _rows;
015         public string TableName
016         {
017             get => _tableName;
018             set
019             {
020                 _tableName = value;
021                 OnPropertyChanged();
022             }
023         }
024         public List<ColumnModel> Columns
025         {
026             get => _columns;
027             set
028             {
029                 _columns = value;
030                 OnPropertyChanged();
031             }
032         }
033         public List<RowModel> Rows
034         {
035             get => _rows;
036             set
037             {
038                 _rows = value;
039                 OnPropertyChanged();
040             }
041         }
042         public event PropertyChangedEventHandler PropertyChanged;
043         protected virtual void OnPropertyChanged([CallerMemberName]
string propertyName = null)
044         {
045             PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
046         }
047     }
048 }
```

Файл Styles.xaml:

```
001 <ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
002
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
003     <Style TargetType="Button">
```

```

004         <Setter Property="Background" Value="#486966"/>
005         <Setter Property="Foreground" Value="White"/>
006         <Setter Property="Padding" Value="12,6"/>
007         <Setter Property="Margin" Value="5"/>
008         <Setter Property="FontWeight" Value="SemiBold"/>
009         <Setter Property="BorderThickness" Value="0"/>
010         <Setter Property="Cursor" Value="Hand"/>
011         <Setter Property="FontSize" Value="12"/>
012         <Setter Property="Opacity" Value="0.9"/>
013         <Setter Property="Template">
014             <Setter.Value>
015                 <ControlTemplate TargetType="Button">
016                     <Border Background="{TemplateBinding
Background}"
017                         BorderBrush="{TemplateBinding
BorderBrush}"
018                         BorderThickness="{TemplateBinding
BorderThickness}"
019                         Padding="12,6"
020                         CornerRadius="2">
021                         <ContentPresenter
HorizontalAlignment="Center"
022                         VerticalAlignment="Center"/>
023                     </Border>
024                 </ControlTemplate>
025             </Setter.Value>
026         </Setter>
027         <Style.Triggers>
028             <Trigger Property="IsMouseOver" Value="True">
029                 <Setter Property="Background" Value="#B0BEC5"/>
030                 <Setter Property="Opacity" Value="1"/>
031             </Trigger>
032             <Trigger Property="IsPressed" Value="True">
033                 <Setter Property="Background" Value="#889C9B"/>
034             </Trigger>
035         </Style.Triggers>
036     </Style>
037     <Style TargetType="ComboBox">
038         <Setter Property="Margin" Value="5"/>
039         <Setter Property="Padding" Value="5"/>
040         <Setter Property="Background" Value="#B2BEBF"/>
041         <Setter Property="BorderBrush" Value="#889C9B"/>
042         <Setter Property="BorderThickness" Value="1"/>
043         <Setter Property="FontSize" Value="14"/>
044     </Style>
045 </ResourceDictionary>

```

Файл BackupService.cs:

```

001 using RepBase.Data;
002 using System;
003 using System.Collections.Generic;
004 using System.Data;
005 using System.Globalization; // Добавляем для CultureInfo
006 using System.IO;
007 using System.Linq;
008 using System.Text;
009 using System.Windows;
010 using Npgsql;
011 using RepBase.Models;
012 using System.Collections.ObjectModel;
013 namespace RepBase.Services

```

```

014 {
015     public class BackupService
016     {
017         private readonly DatabaseManager _databaseManager;
018         private readonly string _backupFolder = "Backups";
019         public BackupService(DatabaseManager databaseManager)
020         {
021             _databaseManager = databaseManager;
022             if (!Directory.Exists(_backupFolder))
023             {
024                 Directory.CreateDirectory(_backupFolder);
025             }
026         }
027         public void CreateBackup(ObservableCollection<TableModel>
tables)
028         {
029             try
030             {
031                 var backupScript =
GenerateDatabaseBackupScript(tables);
032                 SaveBackupScript(backupScript, "database");
033                 MessageBox.Show("Резервная копия базы данных успешно
создана.", "Успех");
034             }
035             catch (Exception ex)
036             {
037                 MessageBox.Show($"Ошибка создания бэкапа:
{ex.Message}", "Ошибка");
038             }
039         }
040         public List<BackupInfo> GetBackups()
041         {
042             var backups = new List<BackupInfo>();
043             var backupFiles = Directory.GetFiles(_backupFolder,
"*.sql").OrderByDescending(f => f);
044             foreach (var file in backupFiles)
045             {
046                 var fileInfo = new FileInfo(file);
047                 backups.Add(new BackupInfo
048                 {
049                     Name = fileInfo.Name,
050                     FilePath = fileInfo.FullName,
051                     CreationTime = fileInfo.CreationTime
052                 });
053             }
054             return backups;
055         }
056         public void RestoreBackup(string backupFilePath)
057         {
058             try
059             {
060                 // Читаем SQL-скрипт из файла
061                 string backupScript =
File.ReadAllText(backupFilePath);
062                 using (var connection =
_databaseManager.GetConnection())
063                 {
064                     connection.Open();
065                     using (var cmd = new NpgsqlCommand(backupScript,
connection))
066                     {
067                         cmd.ExecuteNonQuery();
068                     }
069                 }
070             }
071             catch (Exception ex)
072             {
073                 MessageBox.Show($"Ошибка восстановления бэкапа:
{ex.Message}", "Ошибка");
074             }
075         }
076     }
077 }

```



```

069                }
070                MessageBox.Show("Бэкап успешно восстановлен.",
"Успех");
071            }
072            catch (Exception ex)
073            {
074                throw new Exception($"Ошибка при восстановлении
бэкапа: {ex.Message}");
075            }
076        }
077        public void DeleteBackup(string backupFilePath)
078        {
079            try
080            {
081                if (File.Exists(backupFilePath))
082                {
083                    File.Delete(backupFilePath);
084                    MessageBox.Show("Бэкап успешно удален.",
"Успех");
085                }
086                else
087                {
088                    MessageBox.Show($"Файл бэкапа не найден:
{backupFilePath}", "Ошибка");
089                }
090            }
091            catch (Exception ex)
092            {
093                MessageBox.Show($"Ошибка при удалении бэкапа:
{ex.Message}", "Ошибка");
094            }
095        }
096        private string
GenerateDatabaseBackupScript(ObservableCollection<TableModel> tables)
097        {
098            var script = new StringBuilder();
099            foreach (var table in tables)
100            {
101                var tableData =
_databaseManager.GetTableData(table.TableName);
102                script.AppendLine(GenerateTableBackupScript(table,
tableData));
103                script.AppendLine();
104            }
105            return script.ToString();
106        }
107        private string GenerateTableBackupScript(TableModel table,
DataTable tableData)
108        {
109            var script = new StringBuilder();
110            // Генерируем CREATE TABLE
111            var columnDefs = table.Columns.Select(c =>
112            {
113                string colDef = $"{c.ColumnName}
{MapColumnTypeToSqlType(c.ColumnType)}";
114                if (c.ColumnName.ToLower() == "id") colDef += "
PRIMARY KEY";
115                return colDef;
116            });
117            script.AppendLine($"DROP TABLE IF EXISTS
main.{table.TableName}");
118            script.AppendLine($"CREATE TABLE main.{table.TableName}
({string.Join(", ", columnDefs)});");

```

```

119         // Генерируем INSERT для данных
120         if (tableData != null && tableData.Rows.Count > 0)
121         {
122             foreach (DataRow row in tableData.Rows)
123             {
124                 script.AppendLine(GenerateInsertStatement(table,
row));
125             }
126         }
127         return script.ToString();
128     }
129     private string GenerateInsertStatement(TableModel table,
DataRow row)
130     {
131         var columns = new List<string>();
132         var values = new List<string>();
133         foreach (DataColumn col in row.Table.Columns)
134         {
135             var value = row[col.ColumnName];
136             if (value != DBNull.Value)
137             {
138                 columns.Add(col.ColumnName);
139                 var columnDef = table.Columns.FirstOrDefault(c
=> c.ColumnName == col.ColumnName);
140                 if (columnDef != null)
141                 {
142                     values.Add(FormatValueForSql(value,
columnDef.ColumnType));
143                 }
144             }
145         }
146         if (columns.Any())
147         {
148             return $"INSERT INTO main.{table.TableName}
({string.Join(", ", columns)}) VALUES ({string.Join(", ", values)});";
149         }
150         return "";
151     }
152     private string FormatValueForSql(object value, ColumnType
columnType)
153     {
154         if (value == DBNull.Value) return "NULL";
155         switch (columnType)
156         {
157             case ColumnType.String:
158             case ColumnType.CharacterVarying:
159             case ColumnType.Json:
160                 return $"{value.ToString().Replace("'",
"''")}'";
161             case ColumnType.Boolean:
162                 return (bool)value ? "TRUE" : "FALSE";
163             case ColumnType.DateTime:
164                 return $"{(DateTime)value:yyyy-MM-dd
HH:mm:ss}";
165             case ColumnType.Decimal:
166             case ColumnType.Real:
167                 // Используем InvariantCulture для записи чисел
с точкой
168                 return
Convert.ToDouble(value).ToString(CultureInfo.InvariantCulture);
169             case ColumnType.Integer:
170                 return value.ToString();
171             default:

```

```

172         return value.ToString();
173     }
174 }
175 private string MapColumnTypeToSqlType(ColumnTypes columnType)
176 {
177     switch (columnType)
178     {
179         case ColumnTypes.String:
180             return "TEXT";
181         case ColumnTypes.CharacterVarying:
182             return "VARCHAR(255)";
183         case ColumnTypes.Integer:
184             return "INTEGER";
185         case ColumnTypes.Boolean:
186             return "BOOLEAN";
187         case ColumnTypes.DateTime:
188             return "TIMESTAMP";
189         case ColumnTypes.Decimal:
190             return "DECIMAL";
191         case ColumnTypes.Real:
192             return "REAL";
193         case ColumnTypes.Json:
194             return "JSON";
195         default:
196             throw new ArgumentException($"Unsupported column
type: {columnType}");
197     }
198 }
199 public void SaveBackupScript(string script, string type)
200 {
201     if (!Directory.Exists(_backupFolder))
202     {
203         Directory.CreateDirectory(_backupFolder);
204     }
205     var timestamp = DateTime.Now.ToString("yyyyMMddHHmmss");
206     var fileName =
$"{_backupFolder}/backup_{type}_{timestamp}.sql";
207     File.WriteAllText(fileName, script);
208 }
209 }
210 public class BackupInfo
211 {
212     public string Name { get; set; }
213     public string FilePath { get; set; }
214     public DateTime CreationTime { get; set; }
215 }
216 }

```

Файл ExportService.cs:

```

001 using OfficeOpenXml;
002 using System;
003 using System.Data;
004 using System.Windows;
005 using RepBase.Models;
006 using RepBase.Data;
007 using System.Collections.ObjectModel;
008 using System.IO;
009 using System.Linq;
010 namespace RepBase.Services
011 {
012     public class ExportService
013     {

```

```

014         private readonly DatabaseManager _databaseManager;
015         public ExportService(DatabaseManager databaseManager)
016         {
017             _databaseManager = databaseManager;
018             ExcelPackage.LicenseContext =
OfficeOpenXml.LicenseContext.NonCommercial;
019         }
020         public void ExportCurrentTable(TableModel selectedTable,
DataTable tableData)
021         {
022             if (tableData == null || selectedTable == null)
023             {
024                 MessageBox.Show("Не выбраны данные для экспорта.");
025                 return;
026             }
027             try
028             {
029                 var dialog = new Microsoft.Win32.SaveFileDialog
030                 {
031                     FileName = $"{selectedTable.TableName}_export",
032                     DefaultExt = ".xlsx",
033                     Filter = "Excel Files (*.xlsx)|*.xlsx"
034                 };
035                 if (dialog.ShowDialog() == true)
036                 {
037                     using (var package = new ExcelPackage())
038                     {
039                         var worksheet =
package.Workbook.Worksheets.Add(selectedTable.TableName);
040                         ExportDataTableToWorksheet(tableData,
worksheet);
041                         File.WriteAllBytes(dialog.FileName,
package.GetAsByteArray());
042                         MessageBox.Show($"Данные успешно
экспортированы в {dialog.FileName}");
043                     }
044                 }
045             }
046             catch (Exception ex)
047             {
048                 MessageBox.Show($"Ошибка экспорта в Excel:
{ex.Message}");
049             }
050         }
051         public void ExportScriptResult(string script)
052         {
053             if (string.IsNullOrEmpty(script))
054             {
055                 MessageBox.Show("Введите SQL-скрипт для экспорта
результата.");
056                 return;
057             }
058             try
059             {
060                 var result = _databaseManager.ExecuteQuery(script);
061                 if (result == null || result.Rows.Count == 0)
062                 {
063                     MessageBox.Show("Скрипт не вернул данных для
экспорта.");
064                     return;
065                 }
066                 var dialog = new Microsoft.Win32.SaveFileDialog
067                 {

```

```

068             FileName = "script_result_export",
069             DefaultExt = ".xlsx",
070             Filter = "Excel Files (*.xlsx)|*.xlsx"
071         };
072         if (dialog.ShowDialog() == true)
073         {
074             using (var package = new ExcelPackage())
075             {
076                 var worksheet =
package.Workbook.Worksheets.Add("Script_Result");
077                 ExportDataTableToWorksheet(result,
worksheet);
078                 File.WriteAllBytes(dialog.FileName,
package.GetAsByteArray());
079                 MessageBox.Show($"Результат скрипта успешно
экспортирован в {dialog.FileName}");
080             }
081         }
082     }
083     catch (Exception ex)
084     {
085         MessageBox.Show($"Ошибка экспорта результата
скрипта: {ex.Message}");
086     }
087 }
088 public void
ExportEntireDatabase(ObservableCollection<TableModel> tableItems)
089 {
090     try
091     {
092         var dialog = new Microsoft.Win32.SaveFileDialog
093         {
094             FileName = "database_export",
095             DefaultExt = ".xlsx",
096             Filter = "Excel Files (*.xlsx)|*.xlsx"
097         };
098         if (dialog.ShowDialog() == true)
099         {
100             using (var package = new ExcelPackage())
101             {
102                 foreach (var table in tableItems)
103                 {
104                     var tableData =
_databaseManager.GetTableData(table.TableName);
105                     if (tableData != null &&
tableData.Rows.Count > 0)
106                     {
107                         var worksheet =
package.Workbook.Worksheets.Add(table.TableName);
108                         ExportDataTableToWorksheet(tableData, worksheet);
109                     }
110                 }
111                 if (!package.Workbook.Worksheets.Any())
112                 {
113                     MessageBox.Show("Нет данных для
экспорта.");
114                     return;
115                 }
116                 File.WriteAllBytes(dialog.FileName,
package.GetAsByteArray());
117                 MessageBox.Show($"База данных успешно
экспортирована в {dialog.FileName}");

```

```

118         }
119     }
120 }
121 catch (Exception ex)
122 {
123     MessageBox.Show($"Ошибка экспорта базы данных:
{ex.Message}");
124 }
125 }
126 private void ExportDataTableToWorksheet(DataTable dataTable,
ExcelWorksheet worksheet)
127 {
128     for (int i = 0; i < dataTable.Columns.Count; i++)
129     {
130         worksheet.Cells[1, i + 1].Value =
dataTable.Columns[i].ColumnName;
131         worksheet.Cells[1, i + 1].Style.Font.Bold = true;
132     }
133     for (int row = 0; row < dataTable.Rows.Count; row++)
134     {
135         for (int col = 0; col < dataTable.Columns.Count;
col++)
136         {
137             var value = dataTable.Rows[row][col];
138             worksheet.Cells[row + 2, col + 1].Value = value
== DBNull.Value ? null : value;
139         }
140     }
141     worksheet.Cells[worksheet.Dimension.Address].AutoFitColumns();
142 }
143 }
144 }

```

Файл ScriptService.cs:

```

001 using RepBase.Data;
002 using System;
003 using System.Collections.Generic;
004 using System.Collections.ObjectModel;
005 using System.Data;
006 using System.IO;
007 using System.Linq;
008 using System.Text.Json;
009 using System.Windows;
010 namespace RepBase.Services
011 {
012     public class ScriptService
013     {
014         private readonly DatabaseManager _databaseManager;
015         private readonly Dictionary<string, string> _scripts;
016         private readonly ObservableCollection<string> _scriptNames;
017         public ObservableCollection<string> ScriptNames =>
_scriptNames;
018         public ScriptService(DatabaseManager databaseManager)
019         {
020             _databaseManager = databaseManager;
021             _scripts = new Dictionary<string, string>();
022             _scriptNames = new ObservableCollection<string>();
023             InitializeScripts();
024         }
025         private void InitializeScripts()
026         {

```

```

027          // Загружаем предустановленные скрипты из
default_scripts.json
028          LoadDefaultScripts();
029          // Загружаем пользовательские скрипты из
user_scripts.json
030          LoadUserScripts();
031          // Обновляем коллекцию ScriptNames
032          _scriptNames.Clear();
033          foreach (var scriptName in _scripts.Keys)
034          {
035              _scriptNames.Add(scriptName);
036          }
037      }
038      private void LoadDefaultScripts()
039      {
040          try
041          {
042              if (File.Exists("default_scripts.json"))
043              {
044                  var json =
File.ReadAllText("default_scripts.json");
045                  var defaultScripts =
JsonSerializer.Deserialize<Dictionary<string, string>>(json);
046                  if (defaultScripts != null)
047                  {
048                      foreach (var script in defaultScripts)
049                      {
050                          _scripts[script.Key] = script.Value;
051                      }
052                  }
053              }
054              else
055              {
056                  MessageBox.Show("File 'default_scripts.json' not
found. Default scripts will not be loaded.");
057              }
058          }
059          catch (Exception ex)
060          {
061              MessageBox.Show($"Error loading default scripts:
{ex.Message}");
062          }
063      }
064      private void LoadUserScripts()
065      {
066          try
067          {
068              if (File.Exists("user_scripts.json"))
069              {
070                  var json =
File.ReadAllText("user_scripts.json");
071                  var userScripts =
JsonSerializer.Deserialize<Dictionary<string, string>>(json);
072                  if (userScripts != null)
073                  {
074                      foreach (var script in userScripts)
075                      {
076                          // Пользовательские скрипты
перезаписывают предустановленные
077                          _scripts[script.Key] = script.Value;
078                      }
079                  }
080              }

```

```

081            }
082            catch (Exception ex)
083            {
084                MessageBox.Show($"Error loading user scripts:
{ex.Message}");
085            }
086        }
087        private void SaveScriptsToFile()
088        {
089            try
090            {
091                // Сохраняем только пользовательские скрипты
(исключаем предустановленные)
092                var userScripts = _scripts
093                    .Where(s => !_scripts.ContainsKey(s.Key) ||
!IsDefaultScript(s.Key))
094                    .ToDictionary(s => s.Key, s => s.Value);
095                var json = JsonSerializer.Serialize(userScripts, new
JsonSerializerOptions { WriteIndented = true });
096                File.WriteAllText("user_scripts.json", json);
097            }
098            catch (Exception ex)
099            {
100                MessageBox.Show($"Error saving scripts:
{ex.Message}");
101            }
102        }
103        private bool IsDefaultScript(string scriptName)
104        {
105            // Проверяем, является ли скрипт предустановленным
106            try
107            {
108                if (File.Exists("default_scripts.json"))
109                {
110                    var json =
File.ReadAllText("default_scripts.json");
111                    var defaultScripts =
JsonSerializer.Deserialize<Dictionary<string, string>>(json);
112                    return defaultScripts != null &&
defaultScripts.ContainsKey(scriptName);
113                }
114            }
115            catch
116            {
117                // Если не удалось прочитать default_scripts.json,
считаем, что скрипт не предустановленный
118            }
119            return false;
120        }
121        public string GetScriptContent(string scriptName)
122        {
123            return _scripts.ContainsKey(scriptName) ?
_scripts[scriptName] : "";
124        }
125        public DataTable ExecuteScript(string script)
126        {
127            if (string.IsNullOrEmpty(script))
128            {
129                MessageBox.Show("Введите SQL-скрипт для
выполнения.");
130                return null;
131            }
132            try

```



```

133         {
134             var result = _databaseManager.ExecuteQuery(script);
135             if (result == null || result.Rows.Count == 0)
136             {
137                 MessageBox.Show("Скрипт выполнен, но не вернул
данных.");
138                 return null;
139             }
140             return result;
141         }
142         catch (Exception ex)
143         {
144             MessageBox.Show($"Error executing script:
{ex.Message}");
145             return null;
146         }
147     }
148     public void SaveScript(string script, string scriptName)
149     {
150         if (string.IsNullOrEmpty(script))
151         {
152             MessageBox.Show("Введите SQL-скрипт для
сохранения.");
153             return;
154         }
155         if (_scripts.ContainsKey(scriptName) && scriptName !=
"новый скрипт")
156         {
157             var result = MessageBox.Show($"Скрипт с именем
'{scriptName}' уже существует. Перезаписать?",
158                 "Подтверждение", MessageBoxButtons.YesNo);
159             if (result != DialogResult.Yes)
160             {
161                 return;
162             }
163         }
164         _scripts[scriptName] = script;
165         if (!_scriptNames.Contains(scriptName))
166         {
167             _scriptNames.Add(scriptName);
168         }
169         SaveScriptsToFile();
170         MessageBox.Show($"Скрипт '{scriptName}' успешно
сохранен.");
171     }
172 }
173 }

```

Файл TableService.cs:

```

001 using RepBase.Data;
002 using RepBase.Models;
003 using System;
004 using System.Collections.ObjectModel;
005 using System.Data;
006 using System.Linq;
007 using System.Windows;
008 using Npgsql;
009 using RepBase.ViewModels;
010 using System.Collections.Generic;
011 using System.Text;
012 using System.Globalization; // Добавляем для CultureInfo
013 namespace RepBase.Services

```

```

014 {
015     public class TableService
016     {
017         private readonly DatabaseManager _databaseManager;
018         private readonly BackupService _backupService;
019         public TableService(DatabaseManager databaseManager,
BackupService backupService)
020         {
021             _databaseManager = databaseManager;
022             _backupService = backupService;
023         }
024         public ObservableCollection<TableModel> LoadTables()
025         {
026             var tableItems = new ObservableCollection<TableModel>();
027             try
028             {
029                 var tables = _databaseManager.LoadTables();
030                 foreach (var table in tables)
031                 {
032                     tableItems.Add(table);
033                 }
034             }
035             catch (Exception ex)
036             {
037                 MessageBox.Show($"Error loading tables:
{ex.Message}");
038             }
039             return tableItems;
040         }
041         public DataTable LoadTableData(string tableName)
042         {
043             try
044             {
045                 var dataTable =
_databaseManager.GetTableData(tableName);
046                 CleanDataTable(dataTable);
047                 foreach (DataColumn column in dataTable.Columns)
048                 {
049                     column.ReadOnly = false;
050                 }
051                 return dataTable;
052             }
053             catch (Exception ex)
054             {
055                 MessageBox.Show($"Error loading data for table
{tableName}: {ex.Message}");
056                 return null;
057             }
058         }
059         private void CleanDataTable(DataTable dataTable)
060         {
061             foreach (DataRow row in dataTable.Rows.Cast<DataRow>()
062                 .Where(r => r.ItemArray.All(field => field ==
DBNull.Value)).ToList())
063             {
064                 dataTable.Rows.Remove(row);
065             }
066         }
067         public void AddRow(DataTable tableData, TableModel
selectedTable)
068         {
069             if (selectedTable != null && tableData != null)
070             {

```

```

071             var newRow = tableData.NewRow();
072             if (tableData.Columns.Contains("id"))
073             {
074                 int nextId =
_databaseManager.GetNextId(selectedTable.TableName);
075                 newRow["id"] = nextId;
076             }
077             foreach (DataColumn column in tableData.Columns)
078             {
079                 if (column.ColumnName != "id")
080                 {
081                     newRow[column.ColumnName] = DBNull.Value;
082                 }
083             }
084             tableData.Rows.Add(newRow);
085         }
086     }
087     public void UpdateCell(TableModel selectedTable,
CellUpdateArgs args)
088     {
089         if (selectedTable == null || args == null) return;
090         try
091         {
092             var row = args.Row;
093             var columnName = args.ColumnName;
094             var newValueInput = args.NewValue;
095             object newValue = null;
096             var column = selectedTable.Columns.FirstOrDefault(c
=> c.ColumnName == columnName);
097             if (column == null)
098             {
099                 MessageBox.Show($"Column {columnName} not
found.");
100                 return;
101             }
102             bool isNewRow = row.ItemArray.All(field => field ==
DBNull.Value);
103             if (isNewRow)
104             {
105                 return;
106             }
107             if (!ValidateAndConvertValue(newValueInput,
column.ColumnType, out newValue))
108             {
109                 MessageBox.Show($"Invalid value
'{newValueInput}' for column '{columnName}' of type {column.ColumnType}");
110                 return;
111             }
112             var query = $"UPDATE main.{selectedTable.TableName}
SET {columnName} = @newValue WHERE ";
113             var conditions = new List<string>();
114             var parameters = new List<NpgsqlParameter>
115             {
116                 new NpgsqlParameter("@newValue", newValue ??
DBNull.Value)
117             };
118             int paramCount = 0;
119             foreach (DataColumn col in row.Table.Columns)
120             {
121                 if (col.ColumnName != columnName)
122                 {
123                     var value = row[col.ColumnName];
124                     if (value != DBNull.Value)

```

```

125         {
126             conditions.Add($"{col.ColumnName} =
@p{paramCount}");
127             parameters.Add(new
NpgsqlParameter($"{p{paramCount}", value));
128             paramCount++;
129         }
130     }
131 }
132 query += conditions.Any() ? string.Join(" AND ",
conditions) : "1=1";
133 _databaseManager.ExecuteNonQueryWithParams(query,
parameters);
134     row[columnName] = newValue ?? DBNull.Value;
135 }
136 catch (Exception ex)
137 {
138     MessageBox.Show($"Error updating cell:
{ex.Message}");
139 }
140 }
141 public void SaveNewRow(TableModel selectedTable, DataRowView
rowView)
142 {
143     if (selectedTable == null || rowView == null) return;
144     try
145     {
146         var columns = new List<string>();
147         var values = new List<string>();
148         var parameters = new List<NpgsqlParameter>();
149         int paramCount = 0;
150         if (rowView.Row.Table.Columns.Contains("id"))
151         {
152             columns.Add("id");
153             values.Add("@p0");
154             parameters.Add(new NpgsqlParameter("@p0",
rowView.Row["id"]));
155             paramCount++;
156         }
157         foreach (DataColumn column in
rowView.Row.Table.Columns)
158         {
159             if (column.ColumnName != "id")
160             {
161                 var value = rowView.Row[column.ColumnName];
162                 var columnDef =
selectedTable.Columns.FirstOrDefault(c => c.ColumnName == column.ColumnName);
163                 if (columnDef != null && value !=
DBNull.Value)
164                 {
165                     if
(!ValidateAndConvertValue(value.ToString(), columnDef.ColumnType, out object
validatedValue))
166                     {
167                         MessageBox.Show($"Invalid value
'{value}' for column '{column.ColumnName}' of type {columnDef.ColumnType}");
168                         return;
169                     }
170                     paramCount++;
171                     columns.Add(column.ColumnName);
172                     values.Add($"{p{paramCount}");
173                     parameters.Add(new
NpgsqlParameter($"{p{paramCount}", validatedValue));

```

```

174             }
175         }
176     }
177     if (columns.Any())
178     {
179         var query = $"INSERT INTO
main.{selectedTable.TableName} ({string.Join(", ", columns)}) " +
180             $"VALUES ({string.Join(", ",
values)})";
181
182         _databaseManager.ExecuteNonQueryWithParams(query, parameters);
183     }
184     catch (Exception ex)
185     {
186         MessageBox.Show($"Error saving new row:
{ex.Message}");
187     }
188 }
189 public void DeleteRow(TableModel selectedTable, DataRowView
rowView)
190 {
191     if (selectedTable == null || rowView == null) return;
192     try
193     {
194         // Создаем бэкап строки перед удалением
195         var rowBackupScript =
GenerateRowBackupScript(selectedTable, rowView.Row);
196         _backupService.SaveBackupScript(rowBackupScript,
$"row_{selectedTable.TableName}_{DateTime.Now:yyyyMMddHHmmss}");
197         var whereClause = BuildWhereClause(rowView.Row);
198         _databaseManager.ExecuteNonQuery($"DELETE FROM
main.{selectedTable.TableName} WHERE {whereClause}");
199     }
200     catch (Exception ex)
201     {
202         MessageBox.Show($"Error deleting row:
{ex.Message}");
203     }
204 }
205 public void DeleteTable(TableModel selectedTable,
ObservableCollection<TableModel> tableItems)
206 {
207     if (selectedTable == null)
208     {
209         MessageBox.Show("Выберите таблицу для удаления");
210         return;
211     }
212     var result = MessageBox.Show($"Вы действительно хотите
удалить таблицу '{selectedTable.TableName}'? Это действие будет нельзя
отменить.",
213         "Confirm Delete", MessageBoxButton.YesNo,
MessageBoxImage.Warning);
214     if (result == MessageBoxResult.Yes)
215     {
216         try
217         {
218             // Создаем бэкап таблицы перед удалением
219             var tableData =
_databaseManager.GetTableData(selectedTable.TableName);
220             var tableBackupScript =
GenerateTableBackupScript(selectedTable, tableData);

```

```

221
_backupService.SaveBackupScript(tableBackupScript,
$"table_{selectedTable.TableName}");
222
_databaseManager.DropTable(selectedTable.TableName);
223
                tableItems.Remove(selectedTable);
224
            }
225
            catch (Exception ex)
226
            {
227
                MessageBox.Show($"Ошибка удаления таблицы:
{ex.Message}");
228
            }
229
        }
230
    }
231
    private string BuildWhereClause(DataRow row)
232
    {
233
        var conditions = new StringBuilder();
234
        bool firstCondition = true;
235
        foreach (DataColumn column in row.Table.Columns)
236
        {
237
            var value = row[column.ColumnName];
238
            if (value != DBNull.Value)
239
            {
240
                if (!firstCondition)
241
                {
242
                    conditions.Append(" AND ");
243
                }
244
                conditions.Append($" {column.ColumnName} =
'{value}'");
245
                firstCondition = false;
246
            }
247
        }
248
        return conditions.Length > 0 ? conditions.ToString() :
"1=1";
249
    }
250
    private bool ValidateAndConvertValue(object input,
ColumnType columnType, out object result)
251
    {
252
        result = null;
253
        if (input == null || (input is string str &&
string.IsNullOrEmpty(str)))
254
        {
255
            result = DBNull.Value;
256
            return true;
257
        }
258
        switch (columnType)
259
        {
260
            case ColumnType.Boolean:
261
                if (input is bool boolValue)
262
                {
263
                    result = boolValue;
264
                    return true;
265
                }
266
                if (input is string strValue)
267
                {
268
                    if (bool.TryParse(strValue, out bool
parsedBool))
269
                    {
270
                        result = parsedBool;
271
                        return true;
272
                    }
273
                    if (strValue.ToLower() == "true" || strValue
== "1") { result = true; return true; }

```

```

274             if (strValue.ToLower() == "false" ||
strValue == "0") { result = false; return true; }
275         }
276         return false;
277     case ColumnType.String:
278     case ColumnType.CharacterVarying:
279         result = input.ToString();
280         return true;
281     case ColumnType.Integer:
282         if (int.TryParse(input.ToString(), out int
intValue))
283         {
284             result = intValue;
285             return true;
286         }
287         return false;
288     case ColumnType.DateTime:
289         if (DateTime.TryParse(input.ToString(), out
DateTime dateValue))
290         {
291             result = dateValue;
292             return true;
293         }
294         return false;
295     case ColumnType.Decimal:
296         if (decimal.TryParse(input.ToString(), out
decimal decValue))
297         {
298             result = decValue;
299             return true;
300         }
301         return false;
302     case ColumnType.Real:
303         if (float.TryParse(input.ToString(), out float
floatValue))
304         {
305             result = floatValue;
306             return true;
307         }
308         return false;
309     case ColumnType.Json:
310         try
311         {
312             string inputStr = input.ToString();
313             // Проверяем, является ли строка валидным
JSON
314             System.Text.Json.JsonSerializer.Deserialize<object>(inputStr);
315             result = inputStr;
316             return true;
317         }
318         catch
319         {
320             // Если строка не является валидным JSON,
пытается преобразовать ее
321             string inputStr = input.ToString();
322             try
323             {
324                 // Если это просто строка, оборачиваем
ее в кавычки
325                 if (!inputStr.StartsWith("{") &&
!inputStr.StartsWith("["))
326                 {

```

```

327                                     inputStr = $"\"{inputStr}\"";
328
System.Text.Json.JsonSerializer.Deserialize<object>(inputStr); // Проверяем,
что теперь это валидный JSON
329                                     result = inputStr;
330                                     return true;
331                                     }
332                                     // Если преобразовать не удалось,
возвращаем false
333                                     return false;
334                                     }
335                                     catch
336                                     {
337                                     MessageBox.Show($"Value '{inputStr}' is
not a valid JSON format.");
338                                     return false;
339                                     }
340                                     }
341                                     default:
342                                     return false;
343                                     }
344                                     }
345     private string GenerateRowBackupScript(TableModel table,
DataRow row)
346     {
347         return GenerateInsertStatement(table, row);
348     }
349     private string GenerateTableBackupScript(TableModel table,
DataTable tableData)
350     {
351         var script = new StringBuilder();
352         // Генерируем CREATE TABLE
353         var columnDefs = table.Columns.Select(c =>
354         {
355             string colDef = $"{c.ColumnName}
{MapColumnTypeToSqlType(c.ColumnType)}";
356             if (c.ColumnName.ToLower() == "id") colDef += "
PRIMARY KEY";
357             return colDef;
358         });
359         script.AppendLine($"DROP TABLE IF EXISTS
main.{table.TableName}");
360         script.AppendLine($"CREATE TABLE main.{table.TableName}
({string.Join(", ", columnDefs)}");
361         // Генерируем INSERT для данных
362         if (tableData != null && tableData.Rows.Count > 0)
363         {
364             foreach (DataRow row in tableData.Rows)
365             {
366                 script.AppendLine(GenerateInsertStatement(table,
row));
367             }
368         }
369         return script.ToString();
370     }
371     private string GenerateInsertStatement(TableModel table,
DataRow row)
372     {
373         var columns = new List<string>();
374         var values = new List<string>();
375         foreach (DataColumn col in row.Table.Columns)
376         {
377             var value = row[col.ColumnName];

```



```

378             if (value != DBNull.Value)
379             {
380                 columns.Add(col.ColumnName);
381                 var columnDef = table.Columns.FirstOrDefault(c
=> c.ColumnName == col.ColumnName);
382                 if (columnDef != null)
383                 {
384                     values.Add(FormatValueForSql(value,
columnDef.ColumnType));
385                 }
386             }
387         }
388         if (columns.Any())
389         {
390             return $"INSERT INTO main.{table.TableName}
({string.Join(", ", columns)}) VALUES ({string.Join(", ", values)});";
391         }
392         return "";
393     }
394     private string FormatValueForSql(object value, ColumnType
columnType)
395     {
396         if (value == DBNull.Value) return "NULL";
397         switch (columnType)
398         {
399             case ColumnType.String:
400             case ColumnType.CharacterVarying:
401             case ColumnType.Json:
402                 return $"{value.ToString().Replace("'",
""')}";
403             case ColumnType.Boolean:
404                 return (bool)value ? "TRUE" : "FALSE";
405             case ColumnType.DateTime:
406                 return $"{(DateTime)value:yyyy-MM-dd
HH:mm:ss}";
407             case ColumnType.Decimal:
408             case ColumnType.Real:
409                 // Используем InvariantCulture для записи чисел
с точкой
410                 return
Convert.ToDouble(value).ToString(CultureInfo.InvariantCulture);
411             case ColumnType.Integer:
412                 return value.ToString();
413             default:
414                 return value.ToString();
415         }
416     }
417     private string MapColumnTypeToSqlType(ColumnType columnType)
418     {
419         switch (columnType)
420         {
421             case ColumnType.String:
422                 return "TEXT";
423             case ColumnType.CharacterVarying:
424                 return "VARCHAR(255)";
425             case ColumnType.Integer:
426                 return "INTEGER";
427             case ColumnType.Boolean:
428                 return "BOOLEAN";
429             case ColumnType.DateTime:
430                 return "TIMESTAMP";
431             case ColumnType.Decimal:
432                 return "DECIMAL";

```

```

433             case ColumnType.Real:
434                 return "REAL";
435             case ColumnType.Json:
436                 return "JSON";
437             default:
438                 throw new ArgumentException($"Unsupported column
type: {columnType}");
439         }
440     }
441 }
442 }

```

Файл CreateTableViewModel.cs:

```

001 using System;
002 using System.Collections.Generic;
003 using System.Collections.ObjectModel;
004 using System.ComponentModel;
005 using System.Linq;
006 using System.Runtime.CompilerServices;
007 using System.Windows;
008 using System.Windows.Input;
009 using RepBase.Data;
010 using RepBase.Models;
011 namespace RepBase.ViewModels
012 {
013     public class CreateTableViewModel : INotifyPropertyChanged
014     {
015         private readonly DatabaseManager _databaseManager;
016         private string _tableName;
017         private ObservableCollection<ColumnModel> _columns;
018         public string TableName
019         {
020             get => _tableName;
021             set { _tableName = value; OnPropertyChanged(); }
022         }
023         public ObservableCollection<ColumnModel> Columns
024         {
025             get => _columns;
026             set { _columns = value; OnPropertyChanged(); }
027         }
028         public Array ColumnTypes =>
Enum.GetValues(typeof(ColumnType));
029         public ICommand AddColumnCommand { get; }
030         public ICommand RemoveColumnCommand { get; }
031         public ICommand CreateTableCommand { get; }
032         public CreateTableViewModel(DatabaseManager databaseManager)
033         {
034             _databaseManager = databaseManager;
035             Columns = new ObservableCollection<ColumnModel>();
036             AddColumnCommand = new RelayCommand(AddColumn);
037             RemoveColumnCommand = new RelayCommand(RemoveColumn);
038             CreateTableCommand = new RelayCommand(CreateTable);
039         }
040         private void AddColumn(object parameter)
041         {
042             Columns.Add(new ColumnModel("NewColumn",
ColumnType.String));
043         }
044         private void RemoveColumn(object parameter)
045         {
046             if (parameter is ColumnModel column)
047             {

```

```

048         Columns.Remove(column);
049     }
050 }
051 private void CreateTable(object parameter)
052 {
053     if (string.IsNullOrEmpty(TableName))
054     {
055         MessageBox.Show("Table name cannot be empty.");
056         return;
057     }
058     if (!Columns.Any())
059     {
060         MessageBox.Show("At least one column must be
defined.");
061         return;
062     }
063     // Проверяем, что имена колонок уникальны
064     var columnNames = Columns.Select(c =>
c.ColumnName).ToList();
065     if (columnNames.Distinct().Count() != columnNames.Count)
066     {
067         MessageBox.Show("Column names must be unique.");
068         return;
069     }
070     // Добавляем колон, если его нет
071     if (!Columns.Any(c => c.ColumnName.ToLower() == "id"))
072     {
073         Columns.Insert(0, new ColumnModel("id",
ColumnType.Integer) { ColumnName = "id" });
074     }
075     try
076     {
077         // Формируем определение таблицы
078         var columnDefinitions = new List<string>();
079         foreach (var column in Columns)
080         {
081             string columnDef = $"{column.ColumnName}
{MapColumnTypeToSqlType(column.ColumnType)}";
082             if (column.ColumnName.ToLower() == "id")
083             {
084                 columnDef += " PRIMARY KEY";
085             }
086             columnDefinitions.Add(columnDef);
087         }
088         string tableDefinition = string.Join(", ",
columnDefinitions);
089         _databaseManager.CreateTable(TableName,
tableDefinition);
090         MessageBox.Show($"Table '{TableName}' created
successfully.");
091         (parameter as Window)?.Close();
092     }
093     catch (Exception ex)
094     {
095         MessageBox.Show($"Error creating table:
{ex.Message}");
096     }
097 }
098 private string MapColumnTypeToSqlType(ColumnType columnType)
099 {
100     switch (columnType)
101     {
102         case ColumnType.String:

```

```

103             return "TEXT";
104         case ColumnType.CharacterVarying:
105             return "VARCHAR(255)";
106         case ColumnType.Integer:
107             return "INTEGER";
108         case ColumnType.Boolean:
109             return "BOOLEAN";
110         case ColumnType.DateTime:
111             return "TIMESTAMP";
112         case ColumnType.Decimal:
113             return "DECIMAL";
114         case ColumnType.Real:
115             return "REAL";
116         case ColumnType.Json:
117             return "JSON";
118         default:
119             throw new ArgumentException($"Unsupported column
type: {columnName}");
120     }
121 }
122 public event PropertyChangedEventHandler PropertyChanged;
123 protected virtual void OnPropertyChanged([CallerMemberName]
string propertyName = null)
124 {
125     PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
126 }
127 }
128 }

```

Файл MainViewModel.cs:

```

001 using RepBase.Data;
002 using RepBase.Models;
003 using RepBase.Services;
004 using System;
005 using System.Collections.ObjectModel;
006 using System.ComponentModel;
007 using System.Data;
008 using System.Linq;
009 using System.Runtime.CompilerServices;
010 using System.Windows;
011 using System.Windows.Controls;
012 using System.Windows.Input;
013 namespace RepBase.ViewModels
014 {
015     public class MainViewModel : INotifyPropertyChanged
016     {
017         private readonly DatabaseManager _databaseManager;
018         private readonly TableService _tableService;
019         private readonly ScriptService _scriptService;
020         private readonly BackupService _backupService;
021         private readonly ExportService _exportService;
022         private ObservableCollection<TableModel> _tableItems;
023         private TableModel _selectedTable;
024         private DataTable _tableData;
025         private string _selectedScriptName;
026         private string _currentScript;
027         public ObservableCollection<TableModel> TableItems
028         {
029             get => _tableItems;
030             set { _tableItems = value; OnPropertyChanged(); }
031         }
032     }
033 }

```

```

032         public TableModel SelectedTable
033         {
034             get => _selectedTable;
035             set
036             {
037                 _selectedTable = value;
038                 OnPropertyChanged();
039                 if (value != null)
040                 {
041                     TableData =
042                     _tableService.LoadTableData(value.TableName);
043                 }
044                 else
045                 {
046                     TableData = null;
047                 }
048             }
049         }
050         public DataTable TableData
051         {
052             get => _tableData;
053             set { _tableData = value; OnPropertyChanged(); }
054         }
055         public ObservableCollection<string> ScriptNames =>
056         _scriptService.ScriptNames;
057         public string SelectedScriptName
058         {
059             get => _selectedScriptName;
060             set
061             {
062                 _selectedScriptName = value;
063                 OnPropertyChanged();
064                 UpdateCurrentScript();
065             }
066         }
067         public string CurrentScript
068         {
069             get => _currentScript;
070             set
071             {
072                 _currentScript = value;
073                 Console.WriteLine($"CurrentScript updated to:
074 {value}");
075                 OnPropertyChanged();
076             }
077         }
078         public ICommand SelectTableCommand { get; }
079         public ICommand AddRowCommand { get; }
080         public ICommand DeleteRowCommand { get; }
081         public ICommand UpdateCellCommand { get; }
082         public ICommand SaveNewRowCommand { get; }
083         public ICommand ExportToExcelCommand { get; }
084         public ICommand CreateTableCommand { get; }
085         public ICommand DeleteTableCommand { get; }
086         public ICommand ExecuteScriptCommand { get; }
087         public ICommand SaveScriptCommand { get; }
088         public ICommand ShowExportOptionsCommand { get; }
089         public ICommand CreateBackupCommand { get; }
090         public ICommand ShowRestoreBackupCommand { get; }
091         public MainViewModel()
092         {

```

```

090         _databaseManager = new
DatabaseManager("Host=localhost;Port=5432;Database=rebase;Username=postgres;
Password=postgres");
091         _backupService = new BackupService(_databaseManager);
092         _tableService = new TableService(_databaseManager,
_backupService);
093         _scriptService = new ScriptService(_databaseManager);
094         _exportService = new ExportService(_databaseManager);
095         TableItems = _tableService.LoadTables();
096         SelectTableCommand = new RelayCommand(SelectTable);
097         AddRowCommand = new RelayCommand(AddRow);
098         DeleteRowCommand = new RelayCommand(DeleteRow);
099         UpdateCellCommand = new RelayCommand(UpdateCell);
100         SaveNewRowCommand = new RelayCommand(SaveNewRow);
101         ExportToExcelCommand = new RelayCommand(ExportToExcel);
102         CreateTableCommand = new RelayCommand(CreateTable);
103         DeleteTableCommand = new RelayCommand(DeleteTable);
104         ExecuteScriptCommand = new RelayCommand(ExecuteScript);
105         SaveScriptCommand = new RelayCommand(SaveScript);
106         ShowExportOptionsCommand = new
RelayCommand(ShowExportOptions);
107         CreateBackupCommand = new RelayCommand(CreateBackup);
108         ShowRestoreBackupCommand = new
RelayCommand(ShowRestoreBackup);
109         if (ScriptNames.Contains("новый скрипт"))
110         {
111             SelectedScriptName = "новый скрипт";
112         }
113         else if (ScriptNames.Any())
114         {
115             SelectedScriptName = ScriptNames.First();
116         }
117     }
118     private void SelectTable(object parameter)
119     {
120         if (parameter is TableModel table)
121         {
122             SelectedTable = table;
123         }
124     }
125     private void AddRow(object parameter)
126     {
127         _tableService.AddRow(TableData, SelectedTable);
128         OnPropertyChanged(nameof(TableData));
129     }
130     private void DeleteRow(object parameter)
131     {
132         if (parameter is DataRowView rowView)
133         {
134             _tableService.DeleteRow(SelectedTable, rowView);
135             TableData.Rows.Remove(rowView.Row);
136         }
137     }
138     private void UpdateCell(object parameter)
139     {
140         _tableService.UpdateCell(SelectedTable, parameter as
CellUpdateArgs);
141         OnPropertyChanged(nameof(TableData));
142     }
143     private void SaveNewRow(object parameter)
144     {
145         if (parameter is DataRowView rowView)
146         {

```

```

147         _tableService.SaveNewRow(SelectedTable, rowView);
148         TableData =
_tableService.LoadTableData(SelectedTable.TableName);
149     }
150 }
151 private void ExportToExcel(object parameter)
152 {
153     _exportService.ExportCurrentTable(SelectedTable,
TableData);
154 }
155 private void CreateTable(object parameter)
156 {
157     var createTableWindow = new
CreateTableWindow(_databaseManager)
158     {
159         Owner = Application.Current.MainWindow
160     };
161     createTableWindow.ShowDialog();
162     TableItems = _tableService.LoadTables();
163 }
164 private void DeleteTable(object parameter)
165 {
166     _tableService.DeleteTable(SelectedTable, TableItems);
167     SelectedTable = null;
168     TableData = null;
169     TableItems = _tableService.LoadTables();
170 }
171 private void ExecuteScript(object parameter)
172 {
173     var result =
_scriptService.ExecuteScript(CurrentScript);
174     if (result != null)
175     {
176         TableData = result;
177     }
178 }
179 private void SaveScript(object parameter)
180 {
181     var scriptName = PromptForScriptName();
182     if (!string.IsNullOrEmpty(scriptName))
183     {
184         _scriptService.SaveScript(CurrentScript,
scriptName);
185         SelectedScriptName = scriptName;
186     }
187 }
188 private void ShowExportOptions(object parameter)
189 {
190     var exportOptionsWindow = new ExportOptionsWindow
191     {
192         Owner = Application.Current.MainWindow
193     };
194     if (exportOptionsWindow.ShowDialog() == true)
195     {
196         switch (exportOptionsWindow.SelectedExportType)
197         {
198             case
ExportOptionsWindow.ExportType.CurrentTable:
199                 _exportService.ExportCurrentTable(SelectedTable, TableData);
200                 break;
201             case
ExportOptionsWindow.ExportType.ScriptResult:

```

```

202
_exportService.ExportScriptResult(CurrentScript);
203             break;
204         case
ExportOptionsWindow.ExportType.EntireDatabase:
205
_exportService.ExportEntireDatabase(TableItems);
206             break;
207         }
208     }
209 }
210 private void CreateBackup(object parameter)
211 {
212     _backupService.CreateBackup(TableItems);
213 }
214 private void ShowRestoreBackup(object parameter)
215 {
216     var restoreViewModel = new
RestoreBackupViewModel(_databaseManager);
217     var restoreWindow = new
RestoreBackupWindow(restoreViewModel)
218     {
219         Owner = Application.Current.MainWindow
220     };
221     restoreWindow.ShowDialog();
222     TableItems = _tableService.LoadTables();
223 }
224 public void UpdateCurrentScript()
225 {
226     CurrentScript =
_scriptService.GetScriptContent(SelectedScriptName);
227 }
228 public string GetScriptContent(string scriptName)
229 {
230     return _scriptService.GetScriptContent(scriptName);
231 }
232 private string PromptForScriptName()
233 {
234     var dialog = new ScriptNameDialog();
235     bool? result = dialog.ShowDialog();
236     return result == true &&
!string.IsNullOrEmpty(dialog.ScriptName) ? dialog.ScriptName : null;
237 }
238 public event PropertyChangedEventHandler PropertyChanged;
239 protected virtual void OnPropertyChanged([CallerMemberName]
string propertyName = null)
240 {
241     PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
242 }
243 }
244 public class CellUpdateArgs
245 {
246     public DataRow Row { get; set; }
247     public string ColumnName { get; set; }
248     public object NewValue { get; set; }
249 }
250 public class RelayCommand : ICommand
251 {
252     private readonly Action<object> _execute;
253     private readonly Func<object, bool> _canExecute;
254     public RelayCommand(Action<object> execute, Func<object,
bool> canExecute = null)

```



```

255     {
256         _execute = execute;
257         _canExecute = canExecute;
258     }
259     public event EventHandler CanExecuteChanged
260     {
261         add { CommandManager.RequerySuggested += value; }
262         remove { CommandManager.RequerySuggested -= value; }
263     }
264     public bool CanExecute(object parameter)
265     {
266         return _canExecute == null || _canExecute(parameter);
267     }
268     public void Execute(object parameter)
269     {
270         _execute(parameter);
271     }
272 }
273 }

```

Файл RestoreBackupViewModel.cs:

```

001 using RepBase.Data;
002 using RepBase.Services;
003 using RepBase.ViewModels;
004 using System;
005 using System.Collections.ObjectModel;
006 using System.ComponentModel;
007 using System.Linq;
008 using System.Windows;
009 using System.Windows.Input;
010 namespace RepBase
011 {
012     public class RestoreBackupViewModel : INotifyPropertyChanged
013     {
014         private readonly DatabaseManager _databaseManager;
015         private readonly BackupService _backupService;
016         private ObservableCollection<BackupInfo> _backups;
017         private BackupInfo _selectedBackup;
018         public ObservableCollection<BackupInfo> Backups
019         {
020             get => _backups;
021             set { _backups = value;
OnPropertyChanged(nameof(Backups)); }
022         }
023         public BackupInfo SelectedBackup
024         {
025             get => _selectedBackup;
026             set { _selectedBackup = value;
OnPropertyChanged(nameof(SelectedBackup)); }
027         }
028         public ICommand RestoreBackupCommand { get; }
029         public ICommand DeleteBackupCommand { get; }
030         public RestoreBackupViewModel(DatabaseManager
databaseManager)
031         {
032             _databaseManager = databaseManager;
033             _backupService = new BackupService(_databaseManager);
034             Backups = new
ObservableCollection<BackupInfo>(_backupService.GetBackups());
035             RestoreBackupCommand = new RelayCommand(RestoreBackup);
036             DeleteBackupCommand = new RelayCommand(DeleteBackup);
037         }

```

```

0038         private void RestoreBackup(object parameter)
0039         {
0040             if (SelectedBackup == null)
0041             {
0042                 MessageBox.Show("Пожалуйста, выберите бэкап для
восстановления.");
0043                 return;
0044             }
0045             var result = MessageBox.Show(
0046                 $"{SelectedBackup.Name}? Все текущие данные будут потеряны.",
0047                 "Подтверждение восстановления",
0048                 MessageBoxButton.YesNo,
0049                 MessageBoxImage.Warning);
0050             if (result == MessageBoxResult.Yes)
0051             {
0052                 try
0053                 {
0054                     _backupService.RestoreBackup(SelectedBackup.FilePath);
0055                     MessageBox.Show("Бэкап успешно восстановлен.",
"Успех", MessageBoxButton.OK, MessageBoxImage.Information);
0056                     Application.Current.Windows.OfType<Window>().SingleOrDefault(w =>
w.IsActive)?.Close();
0057                 }
0058                 catch (Exception ex)
0059                 {
0060                     MessageBox.Show($"Ошибка при восстановлении
бэкапа: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
0061                 }
0062             }
0063         }
0064         private void DeleteBackup(object parameter)
0065         {
0066             if (SelectedBackup == null)
0067             {
0068                 MessageBox.Show("Пожалуйста, выберите бэкап для
удаления.");
0069                 return;
0070             }
0071             var result = MessageBox.Show(
0072                 $"{SelectedBackup.Name}? Это действие нельзя отменить.",
0073                 "Подтверждение удаления",
0074                 MessageBoxButton.YesNo,
0075                 MessageBoxImage.Warning);
0076             if (result == MessageBoxResult.Yes)
0077             {
0078                 try
0079                 {
0080                     _backupService.DeleteBackup(SelectedBackup.FilePath);
0081                     Backups.Remove(SelectedBackup);
0082                     SelectedBackup = null;
0083                 }
0084                 catch (Exception ex)
0085                 {
0086                     MessageBox.Show($"Ошибка при удалении бэкапа:
{ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
0087                 }
0088             }
0089         }

```

```
090         public event PropertyChangedEventHandler PropertyChanged;
091         protected virtual void OnPropertyChanged(string
propertyName)
092         {
093             PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs (propertyName));
094         }
095     }
096 }
```

ЗАКЛЮЧЕНИЕ

В рамках данного проекта была разработана прикладная программа RepBase для управления базой данных PostgreSQL, полностью соответствующая заданным требованиям. Приложение предоставляет пользователю удобный графический интерфейс для выполнения операций с базой данных, включая создание, удаление и редактирование таблиц, добавление, редактирование и удаление строк, выполнение SQL-запросов с возможностью их сохранения, экспорт данных в Excel, а также создание и восстановление резервных копий.

Реализованы функции добавления новых таблиц с автоматической генерацией столбца id, удаления таблиц, а также добавления, редактирования и удаления строк. Все операции с данными выполняются через интуитивно понятный интерфейс, включающий список таблиц, таблицу данных и кнопки управления.

Реализован механизм автоматического создания резервных копий перед удалением таблиц и строк, а также возможность создания бэкапа всей базы данных. Функционал восстановления позволяет вернуть данные из бэкапа, минимизируя риск потери информации.

Разработан механизм экспорта данных в Excel, поддерживающий три режима: экспорт текущей таблицы, результатов SQL-запроса и всей базы данных. Экспорт реализован с использованием библиотеки EPPlus, что обеспечивает высокую скорость и удобство работы с файлами Excel.

Разработанное приложение RepBase успешно решает поставленные задачи, предоставляя пользователю мощный и удобный инструмент для управления базой данных PostgreSQL. Реализация всех требуемых функций, продуманный интерфейс и надежный механизм резервного копирования делают приложение готовым к использованию в реальных условиях.