МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Ижевский государственный технический университет имени М. Т. Калашникова»  
Институт «Информатика и вычислительная техника»  
Кафедра «Программное обеспечение»

Работа защищена с оценкой  
«\_\_\_\_\_\_\_\_\_\_»  
Дата \_\_\_\_\_\_\_\_\_\_\_\_  
Подпись\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовой работе  
по дисциплине «Базы данных»  
на тему: «Разработка базы данных для автоматизации Олимпиады»

Выполнил  
студент гр. Б18-191-2 А.В. Морозов

Руководитель

д-р техн. наук, профессор А.Г. Ложкин

Рецензия:  
степень достижения поставленной цели работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
полнота разработки темы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
уровень самостоятельности работы обучающегося\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
недостатки работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ижевск 2021

СОДЕРЖАНИЕ

[1. ПОСТАНОВКА ЗАДАЧИ 3](#_Toc89863570)

[2. ТРЕБОВАНИЯ К БАЗЕ ДАННЫХ И КЛИЕНТСКОМУ ПРИЛОЖЕНИЮ 4](#_Toc89863571)

[3. ЭТАПЫ ПРОЕКТИРОВАНИЯ БАЗЫ ДАННЫХ 5](#_Toc89863572)

[4. ПРОЕКТИРОВАНИЕ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ 9](#_Toc89863573)

[5. ТЕСТОВЫЙ ПРИМЕР 11](#_Toc89863574)

[6. ЗАКЛЮЧЕНИЕ 15](#_Toc89863575)

[7. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 16](#_Toc89863576)

[ПРИЛОЖЕНИЕ 1 17](#_Toc89863577)

[ПРИЛОЖЕНИЕ 2 23](#_Toc89863578)

# ПОСТАНОВКА ЗАДАЧИ

Анализ актуальности темы работы.

В наше время роль автоматизации очень важна. Обойтись без нее невозможно как в учреждении, так и при организации большого мероприятия, например спортивной олимпиады. Обычно под автоматизацией понимается связка из Базы данных и клиентского приложения для нее. При качественном выполнении работ по автоматизации, можно добиться хороших результатов и упростить нашу жизнь.

При организации олимпиады (и для дальнейшего сбора данных при ее проведении) база данных и клиентское приложение для связи с ней просто необходимы, поскольку такая связка помогает не только хранить данные, но и быстро находить и структурировать любую информацию.

Исходя из вышесказанного, считаю данную работу актуальной.

Целями изучения являются:

1. Проектирование и разработки базы данных.

2. Проектирование и разработка клиентского приложения.

Целями закрепления изученных навыков являются:

1. Проектирование и создание базы данных в Microsoft SQL Server (либо иной СУБД по выбору).

2. Разработка клиентского приложения для работы с созданной базой данных (язык программирования любой по выбору).

# ТРЕБОВАНИЯ К БАЗЕ ДАННЫХ И КЛИЕНТСКОМУ ПРИЛОЖЕНИЮ

Требования к базе данных:

1. Не менее 6 таблиц

2. Ограничения: DEFAULT, CHECK, PRIMARY KEY, UNUIQUE, FOREIGN KEY

3. Индексы

4. Проекции (VIEW): по одной таблице, по нескольким таблицам, используя GROUP BY и HAVING,

5. Триггеры выполняющие каскадные изменения данных в связанных таблицах, либо поддерживающие денормализованные данные

Требования к клиентскому приложению:

1. Наличие общего интерфейса, позволяющего работать со всеми таблицами и отчетами

2. При работе с таблицами обеспечивается:

- Перемещение по записям

- Корректировка записей

- Добавление новых записей

- Удаление записей

- Поиск записей по отдельным полям

- Задание фильтра по отдельным полям

- Задание сортировки по отдельным полям

3. Как минимум одна форма должна обеспечивать ввод данных в две таблицы с отношением 1:М

4. Наличие не менее 3 отчетов. Отчет:

- Формируется не менее чем из 2-х таблиц

- Содержит не менее 1-го уровня группировки

- Вычисляемые поля и итоговые данные

- Перед формированием отчета у пользователя запрашиваются фильтры и параметры сортировки по отдельным полям.

# ЭТАПЫ ПРОЕКТИРОВАНИЯ БАЗЫ ДАННЫХ

Выделяют следующие этапы проектирования базы данных:

1) Концептуальное проектирование

2) Логическое проектирование

3) Физическое проектирование

3.1. Концептуальное проектирование

В этап концептуального проектирования входит построение семантической модели предметной области, или информационной модели наиболее высокого уровня абстракции. На данном этапе нет связи с реальной СУБД.

Результатом концептуального проектирования является схема, представленная на рис. 1.

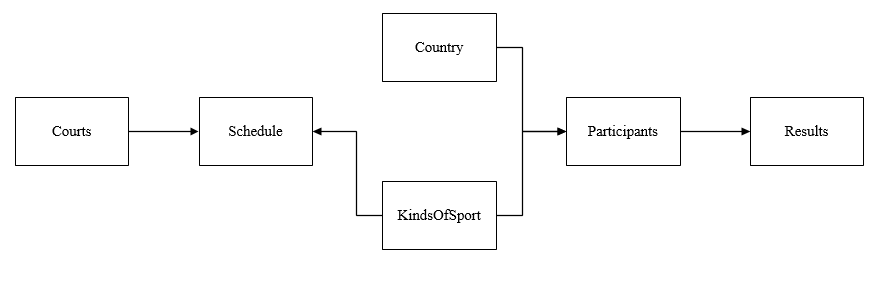


Рис. 1. Семантическая модель предметной области

* 1. Логическое проектирование

На этапе логического проектирования создается полноценная схема базы данных, в данном случае – реляционная модель данных. Для реляционной модели указываются отношения между таблицами, а также внешние ключи таблиц. Логическая схема для реляционной модели представлена на рис. 2. Внешние ключи на схеме изображены с помощью стрелок.

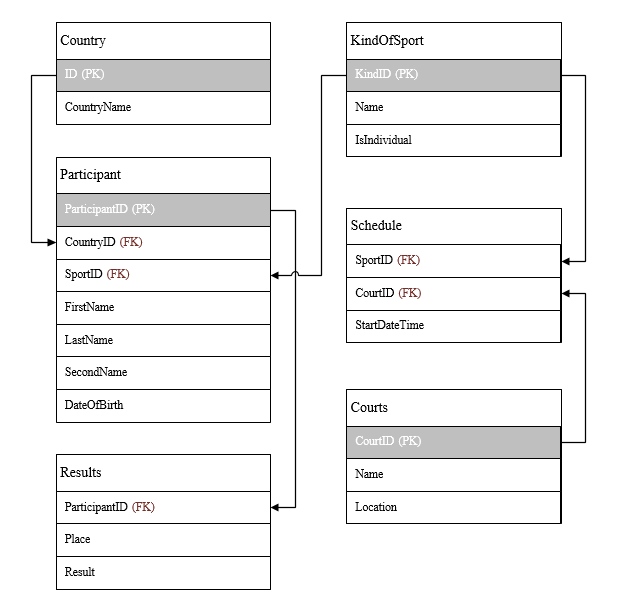


Рис. 2. Логическая схема для реляционной модели

Более подробная информация о таблицах, используемых в текущей работе, представлена в таблице 1.

Таблица 1. Таблицы в БД

|  |  |  |
| --- | --- | --- |
| Наименование таблицы | Поля таблицы | Описание таблицы |
| Country | ID – id страны  Name – название страны | Содержит информацию о странах – участницах в олимпиаде |
| KindOfSport | KindID – id вида спорта  Name – название вида спорта  IsIndividual – является ли вид спорта индивидуальным/командным | Содержит информацию о видах спорта |
| Participants | ParticipantID – id участника  CountruID – id страны  SportID - id вида спорта  FirstName – имя участника  LastName – фамилия участника  SecondName – отчество участника  DateOfBirth – дата рождения участника | Содержит информацию о участниках (спортсменах) |
| Schedule | SportID – id вида спорта  CourtID – id площадки  StartDateTime – дата и время начала | Содержит информацию о расписании стартов на олимпиаде |
| Court | CourtID – id  Name – название площадки  Location – расположение площадки | Содержит информацию о спортивных площадках |
| Results | ParticipantID – id участника  Place – место, занятое участником  Result – результат | Содержит результаты спортсменов |

На этапе физического проектирования создается схема базы данных для конкретно выбранной СУБД. В данном случае это T-SQL. Этап включает в себя создание ограничений, индексов, триггеров, представлений и т.д. В конечном итоге результатом выполнения данного этапа является скрипт, физически создающий базу данных. Результат выполнения этапа представлен в приложении 1.

# ПРОЕКТИРОВАНИЕ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

Проектирование клиентского приложения состоит из следующих этапов:

1) Выбор платформы и языка программирования

2) Процедурная декомпозиция

3) Физическое проектирование (разработка приложения)

4.1. Выбор платформы и языка программирования

В качестве языка программирования и платформы выбран .NET Framework и язык C#. Платформа является быстрой и эффективной, она включает в себя все необходимые инструменты для создания клиентского приложения с нуля.

* 1. Процедурная декомпозиция

Процедурная декомпозиция это представление разрабатываемой программы в виде подпрограмм, совокупность которых выполняет поставленную задачу.

Язык C# - объектно-ориентированный, поэтому для решения большинства конкретных подзадач создаются классы. Для более мелких подзадач, создаются методы внутри класса. Процедурная декомпозиция осуществлена посредством создания структурной схемы программы. Она представлена на рис. 3 на следующей странице. Стоит отметить, что для оптимизации клиентского приложения, было принято решение, свести все к универсальным методам, для повторного их использования. Это позволяет уменьшить количество кода и повысить его качество.

Также при проектировании использовался паттерн проектирования – репозиторий, который покрывает всю работу с БД.

* 1. Физическое проектирование

Физическое проектирование - это этап написания кода, выполняющего поставленную задачу. Текст программы на языке C# представлен в приложении 2.

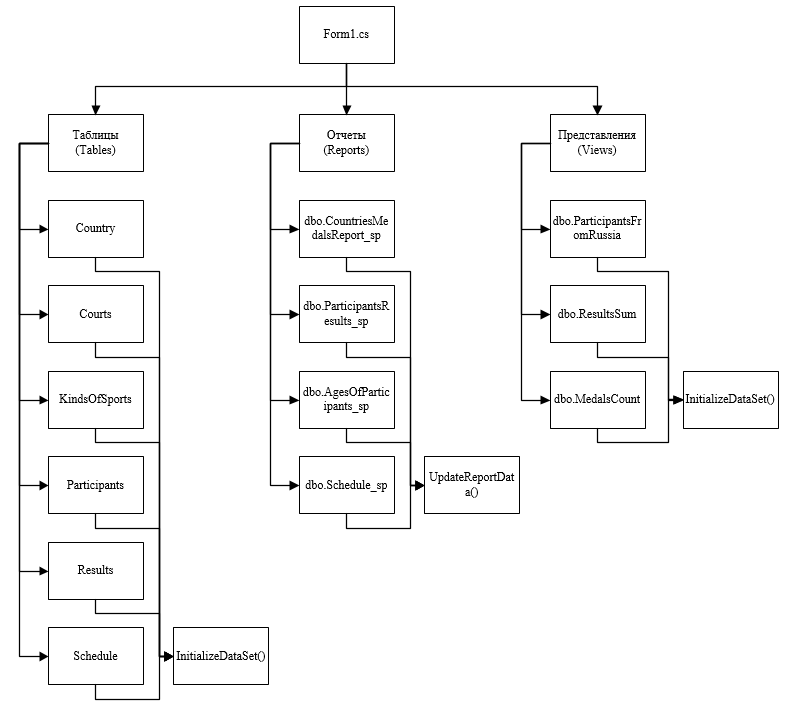


Рис. 3. Структурная схема программы

# ТЕСТОВЫЙ ПРИМЕР

При открытии приложения появляется пустое окно. Слева представлена древовидная схема объектов Базы данных, справа находится DataGridView, для отображения и редактирования данных в БД.



Рис. 4. Стартовое окно

Для навигации между объектами, нужно раскрыть вкладку, и выбрать нужный нам объект.

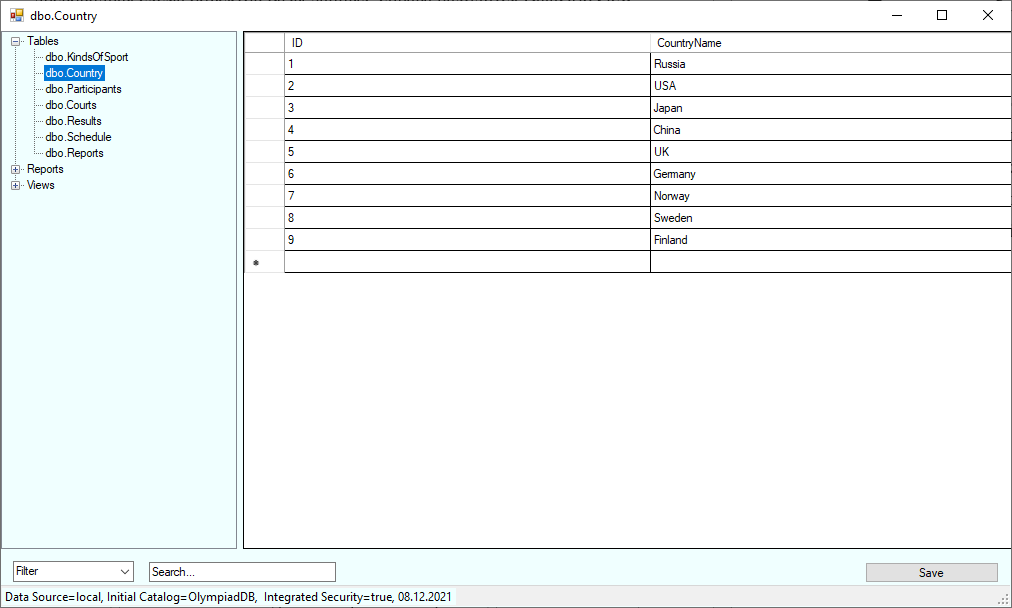


Рис. 5. Таблица стран

Добавим данные в таблицу стран. Для этого Заполним пустую строчку в таблице и нажмем кнопку Save.

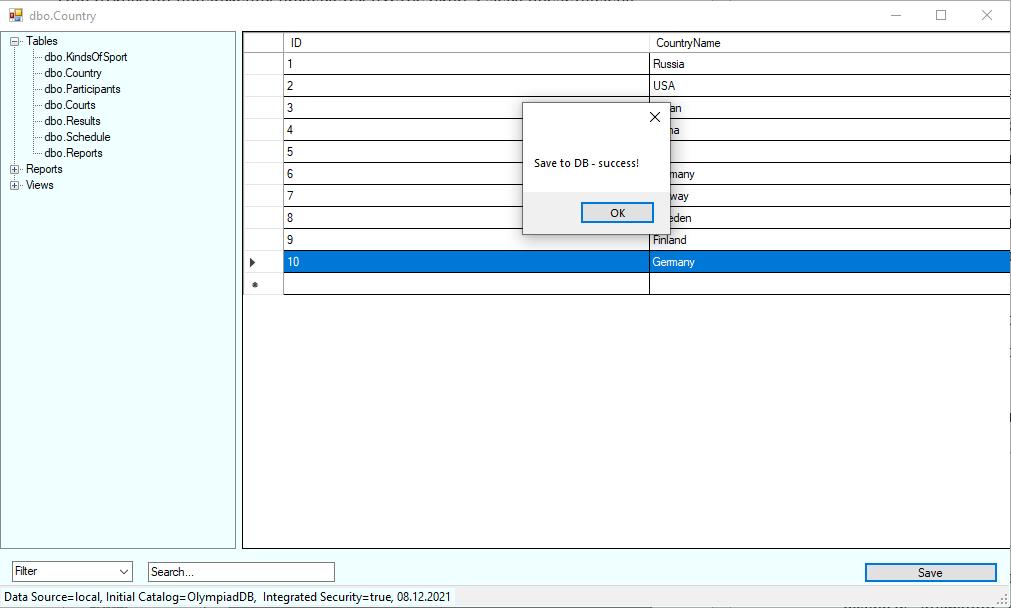


Рис. 6. Сохранение таблицы

Для других операций (удаление, редактирование существующих записей) алгоритм сохраняется. В случае ошибки или неправильного ввода данных, строка с ошибкой будет подсвечена.

Для фильтрации и поиска данных используются TextBox’ы внизу приложения, в первом можно выбрать колонку для фильтрации, во втором осуществляется сам поиск.

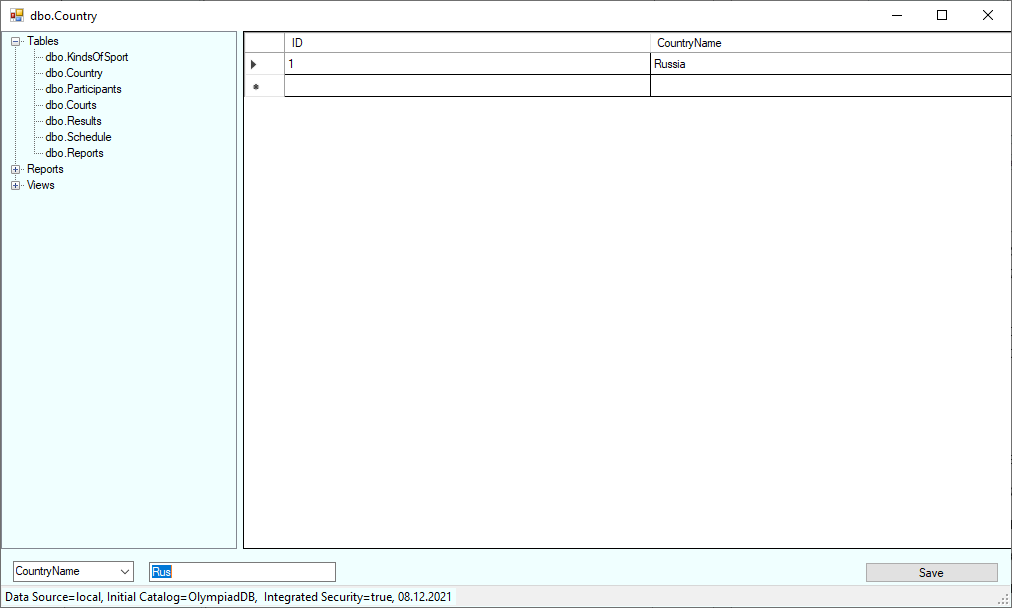


Рис. 7. Поиск по колонке CountryName

Для работы с отчетами, нужно выбрать любой отчет из вкладки Reports. Откроется новое окно, с параметрами отчета, кнопкой экспорта в Pdf, и сами отчетом. Сгенерируем отчет – Расписание на выбранную дату (Schedule on date)

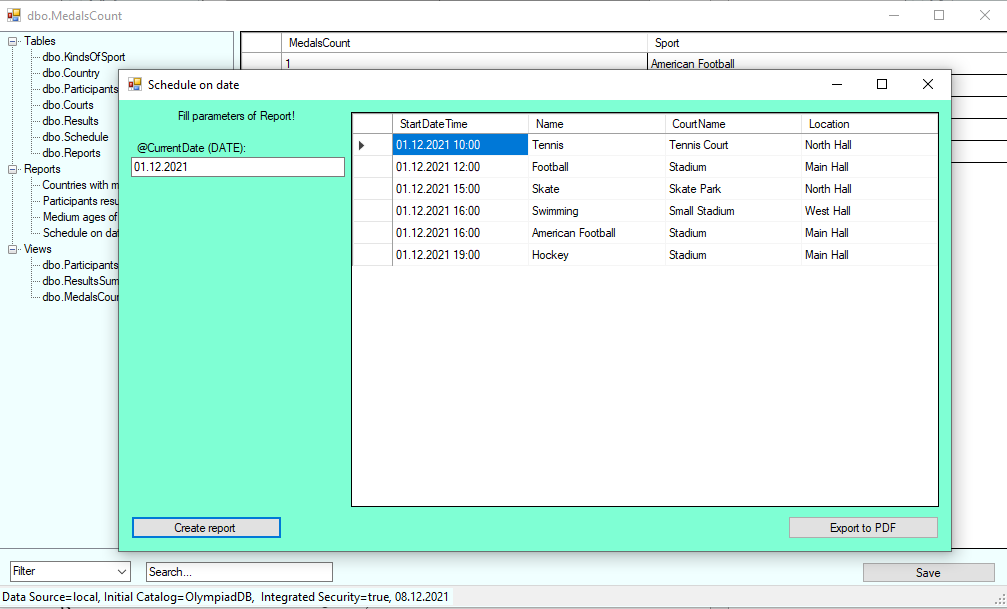


Рис. 8. Отчет Schedule on date



Рис. 9. Пример экспортированного отчета

Во всех отчетах параметры не являются обязательными, они лишь сужают выборку необходимых данных. Поэтому можно не заполнять параметры ни у одного отчета.

Работа с Представлениями (View) ничем не отличается от работы с таблицы, за исключением того, что представление нельзя ни отредактировать, ни сохранить. Просмотр представления показан на рисунке 10 на следующей странице.

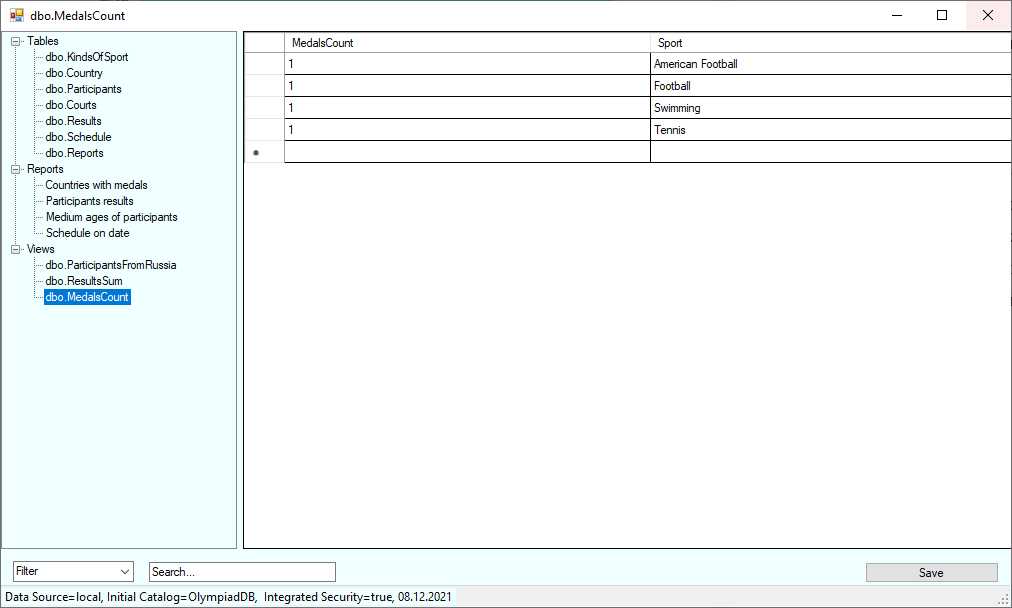


Рис. 10. Представление

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы произведена автоматизация проведения большого мероприятия - Олимпиады.

Подробно изучены, описаны и реализованы этапы проектирования базы данных, создана семантическая модель, создана реляционная модель, осуществлено физическое проектирование базы данных посредством написания скриптов в СУБД T-SQL (Microsoft SQL).

Подробно изучены, описаны и реализованы этапы проектирования клиентского приложения. Произведена процедурная декомпозиция посредством создания структурной схемы программы. Написано приложение для взаимодействия с базой данных на языке программирования C# с использованием технологии .NET Framework.

Программа и база данных могут использоваться для автоматизации Олимпиады - для хранения, извлечения, добавления и редактирования находящейся в базе данных информации, а также для обобщения данных и сбора статистики с помощью формирования отчетов. Кроме того, клиентское приложение можно подключить к любой базе данных T-SQL, для проведения базовых операций с данными, за исключением генерации отчетов (при небольшом расширении БД, можно добавить поддержку отчетов в любую базу). Цели курсовой работы достигнуты.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дейт, К. Дж. Введение в системы баз данных, 8-е издание: Пер. с англ. – М.: Издательский дом «Вильямс», 2005. – 1328 с.: ил. – Парал. тит. англ.
2. Рихтер, CLR via C# : Пер. с англ. – СПб. : Издательство «Питер», 2017. – 896 с. : ил. – Парал. тит. англ.
3. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования – СПб : Издательский дом «Питер», 2001. – 368 с. : ил.
4. Станек Уильям Р. Microsoft SQL Server 2005. Справочник администратора : Пер. с англ. – М.: Издательство «Русская редакция», 2008. – 554 с.: ил.
5. Бен-Ган Ицик, Microsoft SQL Server 2012: Пер. с англ. : Уч. Пос. – М. : Издательство «Эксмо», 2015. - 400 с. : ил. – Парал. тит. англ.
6. Справочное руководство по T-SQL // docs.microsoft.com, [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/sql/t-sql (Дата обращения: 01.12.2021)

# ПРИЛОЖЕНИЕ 1

СТРУКТУРА БАЗЫ ДАННЫХ

Создание таблицы Courts

IF OBJECT\_ID('dbo.Courts') IS NULL

BEGIN

CREATE TABLE [dbo].[Courts]

(

[CourtID] INT NOT NULL IDENTITY(1,1),

[Name] NVARCHAR(64) NOT NULL UNIQUE,

[Location] NVARCHAR(256) NOT NULL,

PRIMARY KEY (CourtID)

)

END;

GO

INSERT INTO [dbo].[Courts] (Name, Location)

VALUES

('Swimming Pool', 'Main Hall'),

('Tennis Court', 'North Hall'),

('Skate Park', 'North Hall'),

('Stadium', 'Main Hall'),

('Small Stadium', 'West Hall')

Создание таблицы Country

IF OBJECT\_ID('dbo.Country') IS NULL

BEGIN

CREATE TABLE [dbo].[Country]

(

[ID] INT NOT NULL IDENTITY(1,1),

[CountryName] NVARCHAR(64) NOT NULL,

PRIMARY KEY (ID)

)

END;

GO

INSERT INTO [dbo].[Country] (CountryName)

VALUES

('Russia'),

('USA'),

('Japan'),

('China'),

('UK'),

('Germany'),

('Norway'),

('Sweden'),

('Finland')

Создание таблицы KindsOfSport с индексом

IF OBJECT\_ID('dbo.KindsOfSport') IS NULL

BEGIN

CREATE TABLE [dbo].[KindsOfSport]

(

[KindID] INT NOT NULL IDENTITY(1,1),

[Name] NVARCHAR(64) NOT NULL UNIQUE,

[IsIndividual] BIT NOT NULL,

PRIMARY KEY (KindID)

)

END;

GO

INSERT INTO [dbo].[KindsOfSport] (Name, IsIndividual)

VALUES

('Tennis', 1),

('Football', 0),

('Skate', 1),

('Swimming', 1),

('American Football', 0),

('Hockey', 0)

GO

CREATE INDEX index\_KindsOfSport\_Name

ON dbo.KindsOfSport([Name]);

Создание таблицы Participants с индексом

IF OBJECT\_ID('dbo.Participants') IS NULL

BEGIN

CREATE TABLE [dbo].[Participants]

(

[CountryID] INT,

[SportID] INT,

[ParticipantID] INT IDENTITY(100,1),

[FirstName] NVARCHAR(30) NOT NULL,

[LastName] NVARCHAR(30) NOT NULL,

[SecondName] NVARCHAR(30) DEFAULT '',

[DateOfBirth] DATETIME NOT NULL,

PRIMARY KEY (ParticipantID),

FOREIGN KEY (SportID) REFERENCES KindsOfSport(KindID),

FOREIGN KEY (CountryID) REFERENCES Country(ID)

)

END;

GO

INSERT INTO [dbo].[Participants] (CountryID, SportID, FirstName, LastName, SecondName, DateOfBirth)

VALUES

( 1, 1, 'Andrei', 'Morozov', 'Vladimirovich', '04.29.2000'),

( 2, 2, 'Evgenii', 'Lad', '', '05.05.1995'),

( 3, 1, 'Mark', 'Forest', 'Chris', '12.01.1998'),

( 4, 5, 'Chris', 'Lamb', '', '07.10.1999'),

( 5, 4, 'Felix', 'Chest', '', '12.12.2001'),

( 6, 6, 'Andrej', 'Cheese', '', '02.28.2001'),

( 7, 4, 'Emily', 'Cheese', '', '03.04.2000'),

( 8, 1, 'Taylor', 'Swift', '', '03.04.2000'),

( 9, 2, 'Bri', 'Tales', '', '08.09.2000'),

( 1, 4, 'Kate', 'Karenina', '', '05.12.2000')

GO

CREATE INDEX index\_Participants\_FirstName

ON dbo.Participants(FirstName);

Создание таблицы Schedule

IF OBJECT\_ID('dbo.Schedule') IS NULL

BEGIN

CREATE TABLE [dbo].[Schedule]

(

[SportID] INT,

[CourtID] INT,

[StartDateTime] DATETIME NOT NULL,

FOREIGN KEY (SportID) REFERENCES KindsOfSport(KindID),

FOREIGN KEY (CourtID) REFERENCES Courts(CourtID)

)

END;

GO

INSERT INTO dbo.Schedule (SportID, CourtID, StartDateTime)

VALUES

(1, 2, '2021-12-01 10:00:00'),

(2, 4, '2021-12-01 12:00:00'),

(3, 3, '2021-12-01 15:00:00'),

(4, 5, '2021-12-01 16:00:00'),

(5, 4, '2021-12-01 16:00:00'),

(6, 4, '2021-12-01 19:00:00')

Создание таблицы Results

IF OBJECT\_ID('dbo.Results') IS NOT NULL

BEGIN

DROP TABLE [dbo].[Results]

END

GO

IF OBJECT\_ID('dbo.Results') IS NULL

BEGIN

CREATE TABLE [dbo].[Results]

(

[ParticipantID] INT,

[Place] SMALLINT NOT NULL,

[Result] NVARCHAR(256) NOT NULL,

CHECK (Place > 0),

FOREIGN KEY (ParticipantID) REFERENCES Participants(ParticipantID)

)

END;

GO

INSERT INTO [dbo].[Results] (ParticipantID, Place, Result)

VALUES

(100, 1, RAND(6)),

(101, 2, RAND(6)),

(102, 4, RAND(6)),

(103, 1, RAND(6)),

(104, 6, RAND(6)),

(105, 1, RAND(6)),

(106, 2, RAND(6))

Создание представления MedalsCount

DROP VIEW [MedalsCount]

GO

CREATE VIEW [MedalsCount] AS

SELECT COUNT(\*) as MedalsCount, K.[Name] as Sport

FROM [dbo].[Results] as R

INNER JOIN [dbo].[Participants] as P on P.ParticipantID = R.ParticipantID

INNER JOIN [dbo].[KindsOfSport] as K on K.KindID = P.SportID

GROUP BY K.Name

Создание представления ParticipantsFromRussia

DROP VIEW [ParticipantsFromRussia]

GO

CREATE VIEW [ParticipantsFromRussia] AS

SELECT \*

FROM [dbo].[Participants]

WHERE [CountryID] = 1;

Создание представления ResultsView

DROP VIEW [ResultsSum]

GO

CREATE VIEW [ResultsSum] AS

SELECT [Place], [Result], [CountryName], K.[Name] as Sport, [FirstName] + ' ' + [LastName] as Participant

FROM [dbo].[Results] as R

INNER JOIN [dbo].[Participants] as P on P.ParticipantID = R.ParticipantID

INNER JOIN [dbo].[Country] as C on C.ID = P.CountryID

INNER JOIN [dbo].[KindsOfSport] as K on K.KindID = P.SportID

Создание триггера для таблицы Country

CREATE TRIGGER CountryDisqualification

ON [dbo].[Country]

INSTEAD OF DELETE

AS

BEGIN

DELETE FROM [dbo].[Results]

WHERE [ParticipantID] in

(

SELECT P.[ParticipantID]

FROM DELETED as D

INNER JOIN [dbo].[Participants] as P on P.[CountryID] = D.[ID]

)

DELETE FROM [dbo].[Participants]

WHERE [ParticipantID] in

(

SELECT P.[ParticipantID]

FROM DELETED as D

INNER JOIN [dbo].[Participants] as P on P.[CountryID] = D.[ID]

)

DELETE FROM [dbo].[Country]

WHERE [ID] in (SELECT ID FROM DELETED)

END

Создание триггера для таблицы Participants

ON [dbo].[Participants]

INSTEAD OF DELETE

AS

BEGIN

DELETE FROM [dbo].[Results]

WHERE [ParticipantID] in (SELECT [ParticipantID] FROM DELETED)

DELETE FROM [dbo].[Participants]

WHERE [ParticipantID] in (SELECT [ParticipantID] FROM DELETED)

END

Создание хранимой процедуры для отчета - Medium ages of participants

IF OBJECT\_ID('dbo.AgesOfParticipants\_sp') IS NOT NULL

BEGIN

DROP PROCEDURE dbo.AgesOfParticipants\_sp

END

GO

CREATE PROCEDURE dbo.AgesOfParticipants\_sp

@MinAge SMALLINT = NULL,

@MaxAge SMALLINT = NULL

AS

BEGIN

DECLARE @Today DATETIME = GETDATE();

SELECT

[p].[ParticipantID],

[p].[FirstName] + ' ' + [p].[LastName] AS [Name],

DATEDIFF(year, [p].[DateOfBirth], @Today) AS [Years]

FROM

[dbo].[Participants] as [p]

WHERE

(@MinAge IS NULL OR DATEDIFF(year, [p].[DateOfBirth], @Today) > @MinAge)

AND (@MaxAge IS NULL OR DATEDIFF(year, [p].[DateOfBirth], @Today) < @MaxAge)

END

Создание хранимой процедуры для отчета - Participants results

IF OBJECT\_ID('dbo.ParticipantsResults\_sp') IS NOT NULL

BEGIN

DROP PROCEDURE dbo.ParticipantsResults\_sp

END

GO

CREATE PROCEDURE dbo.ParticipantsResults\_sp

@Country NVARCHAR(64) = NULL,

@Sport NVARCHAR(64) = NULL

AS

BEGIN

SELECT

[r].[Place],

[r].[Result],

[p].[ParticipantID],

[p].[FirstName] + ' ' + [p].[LastName] AS [Name],

[k].[Name] AS [Sport],

[c].[CountryName]

FROM

[dbo].[Results] AS [r]

INNER JOIN [dbo].[Participants] AS [p] ON [r].[ParticipantID] = [p].[ParticipantID]

INNER JOIN [dbo].[KindsOfSport] AS [k] ON [p].[SportID] = [k].[KindID]

INNER JOIN [dbo].[Country] AS [c] ON [p].[CountryID] = [c].[ID]

WHERE

(@Country IS NULL OR [c].CountryName = @Country)

AND (@Sport IS NULL OR [k].[Name] = @Sport)

END

Создание хранимой процедуры для отчета - Countries with medals

IF OBJECT\_ID('dbo.CountriesMedalsReport\_sp') IS NOT NULL

BEGIN

DROP PROCEDURE dbo.CountriesMedalsReport\_sp

END

GO

CREATE PROCEDURE dbo.CountriesMedalsReport\_sp

@ListOfCountries NVARCHAR(500) = 'ALL'

AS

BEGIN

SELECT

[c].[CountryName],

COUNT([r].[Place]) as [MedalsCount]

FROM

[dbo].[Country] AS [c]

INNER JOIN [dbo].[Participants] AS [p] ON [c].ID = [p].[CountryID]

INNER JOIN [dbo].[Results] AS [r] ON [p].[ParticipantID] = [r].[ParticipantID]

WHERE

@ListOfCountries = 'ALL'

OR [c].[CountryName] in (SELECT value FROM STRING\_SPLIT(@ListOfCountries, ','))

GROUP BY

[c].[CountryName]

END

GO

Создание хранимой процедуры для отчета - Schedule on date

IF OBJECT\_ID('dbo.Schedule\_sp') IS NOT NULL

BEGIN

DROP PROCEDURE dbo.Schedule\_sp

END

GO

CREATE PROCEDURE dbo.Schedule\_sp

@CurrentDate DATE = NULL

AS

BEGIN

SELECT

[s].[StartDateTime],

[k].[Name],

[c].[Name] as [CourtName],

[c].[Location]

FROM

[dbo].[Schedule] AS [s]

INNER JOIN [dbo].[KindsOfSport] AS [k] ON [s].[SportID] = [k].[KindID]

INNER JOIN [dbo].[Courts] AS [c] ON [c].[CourtID] = [s].[CourtID]

WHERE

CAST([s].[StartDateTime] AS DATE) = @CurrentDate

OR @CurrentDate IS NULL

END

# ПРИЛОЖЕНИЕ 2

ТЕКСТ ПРОГРАММЫ

Класс Form1:

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using WindowsFormsApp1.Classes;

using WindowsFormsApp1.Forms;

namespace WindowsFormsApp1

{

public partial class Form1 : Form

{

private App app = new App();

DataSet dataset = new DataSet();

string currentTable;

public Form1()

{

InitializeComponent();

}

protected override void OnLoad(EventArgs e)

{

base.OnLoad(e);

FillStatusStrip();

InitializeTree();

}

private void FillStatusStrip()

{

var split = SqlHelper.connString.Split(';');

var builder = new StringBuilder();

builder.Append(split[0].EndsWith(".") ? split[0].Replace(".", "local") : split[0]);

builder.Append(", ");

builder.Append(split[1]);

builder.Append(", ");

builder.Append(split[2]);

builder.Append(", ");

builder.Append(DateTime.Today.ToShortDateString());

toolStripStatusLabel1.Text = builder.ToString();

}

private void InitializeTree()

{

var rootItems = SqlHelper.GetTables(SqlScripts.SelectTables);

var rootItemsForReports = SqlHelper.GetReports(SqlScripts.SelectReports);

var rootItemsForViews = SqlHelper.GetViews(SqlScripts.SelectViews);

try

{

treeView1.BeginUpdate();

treeView1.Nodes.Add(CreateNodesForTables(rootItems));

treeView1.Nodes.Add(CreateNodesForReports(rootItemsForReports));

treeView1.Nodes.Add(CreateNodesForViews(rootItemsForViews));

}

finally

{

treeView1.EndUpdate();

};

}

private TreeNode CreateNodesForViews(IList<string> items)

{

var node = new TreeNode() { Text = "Views" };

node.Tag = 0;

foreach (var item in items)

{

var child = new TreeNode() { Text = item };

node.Nodes.Add(child);

}

return node;

}

private TreeNode CreateNodesForTables(IList<string> items)

{

var node = new TreeNode() { Text = "Tables" };

node.Tag = 0;

foreach (var item in items)

{

var child = new TreeNode() { Text = item };

node.Nodes.Add(child);

}

return node;

}

private TreeNode CreateNodesForReports(IList<string> items)

{

var node = new TreeNode() { Text = "Reports" };

node.Tag = 1;

foreach (var item in items)

{

var child = new TreeNode() { Text = item };

node.Nodes.Add(child);

}

return node;

}

private void treeView1\_NodeMouseClick(object sender, TreeNodeMouseClickEventArgs e)

{

comboBox1.SelectedIndex = -1;

comboBox1.Text="Filter";

textBox1.Clear();

textBox1.Text = "Search...";

if (e.Node.Parent?.ToString() == "TreeNode: Tables" || e.Node.Parent?.ToString() == "TreeNode: Views")

{

currentTable = e.Node.Text;

InitializeDataSet(e.Node.Text, e.Node.Parent?.ToString() == "TreeNode: Tables" ? false : true);

}

InitializeCombo();

if (e.Node.Parent?.ToString() == "TreeNode: Reports")

{

Form reportForm = new ReportFor\_m(e.Node.Text);

reportForm.TopMost = true;

reportForm.Show();

}

}

private void InitializeCombo()

{

dataGridView1.CurrentCell = null;

var ds = (DataSet)dataGridView1?.DataSource;

string[] columnNames = ds?.Tables[0]?.Columns.Cast<DataColumn>()

.Select(x => x.ColumnName)

.ToArray();

if (columnNames != null)

{

comboBox1.Items.Clear();

comboBox1.Items.AddRange(columnNames);

}

}

private void InitializeDataSet(string tableName, bool isView = false)

{

dataset = SqlHelper.GetTableDataSet(tableName);

this.Text = tableName;

dataGridView1.DataSource = null;

dataGridView1.AutoGenerateColumns = true;

dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

dataGridView1.DataSource = dataset;

dataGridView1.DataMember = dataset.Tables[0].TableName;

dataGridView1.ReadOnly = isView;

}

private void button1\_Click(object sender, EventArgs e)

{

if (dataGridView1?.DataSource != null)

{

DataSet ds = (DataSet)dataGridView1.DataSource;

if (ds.HasChanges())

{

try

{

SqlHelper.SaveAndCommitToDb((DataSet)dataGridView1.DataSource, currentTable);

app.LogSuccess("Save to DB");

}

catch (SqlException ex)

{

app.LogError(ex);

}

}

if (dataGridView1.CurrentRow?.Index != default)

{

var temp = dataGridView1.CurrentRow.Index;

InitializeDataSet(currentTable);

dataGridView1.FirstDisplayedScrollingRowIndex = temp;

dataGridView1.Rows[temp].Selected = true;

}

}

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

string searchValue = textBox1.Text;

string searchRowName = comboBox1.Text;

foreach (DataGridViewRow row in dataGridView1.Rows)

{

int rowIndex = row.Index;

dataGridView1.Rows[rowIndex].Visible = true;

}

if (dataGridView1 != null && dataGridView1?.DataSource != null)

{

dataGridView1.FirstDisplayedScrollingRowIndex = 0;

dataGridView1.SelectionMode = DataGridViewSelectionMode.FullRowSelect;

try

{

foreach (DataGridViewRow row in dataGridView1.Rows)

{

for (int i = 0; i < row.Cells.Count; i++)

{

if (dataGridView1.Columns[row.Cells[i].ColumnIndex].HeaderText == searchRowName)

{

if (row.Cells[i].Value == null || !(row.Cells[i].Value.ToString().ToUpper().StartsWith(searchValue.ToUpper())))

{

int rowIndex = row.Index;

dataGridView1.Rows[rowIndex].Visible = false;

break;

}

}

}

if (row.Index == dataGridView1.Rows.Count - 2)

break;

}

}

catch (Exception exc)

{

MessageBox.Show(exc.Message);

}

}

}

private void textBox1\_Click(object sender, EventArgs e)

{

if(textBox1.Text == "Search...")

textBox1.Clear();

}

private void comboBox1\_MouseClick(object sender, MouseEventArgs e)

{

comboBox1.Text = "";

}

}

}

Класс ReportForm:

using iTextSharp.text;

using iTextSharp.text.pdf;

using System;

using System.Data;

using System.Data.SqlClient;

using System.IO;

using System.Reflection;

using System.Windows.Forms;

using WindowsFormsApp1.Classes;

namespace WindowsFormsApp1.Forms

{

public partial class ReportFor\_m : Form

{

private readonly App app = new App();

private string currentReport;

private int paramsCount = 0;

public ReportFor\_m(string report)

{

InitializeComponent();

currentReport = report;

this.Text = currentReport;

paramsCount = 0;

HideParamsBoxes();

UpdateReportData();

}

private void HideParamsBoxes()

{

textBoxParam1.Visible = false;

textBoxParam2.Visible = false;

textBoxParam3.Visible = false;

paramLabel1.Visible = false;

paramLabel2.Visible = false;

paramLabel3.Visible = false;

}

private void UpdateReportData()

{

var paramSP = GetReportParams().Tables[0].Rows;

switch (paramSP.Count)

{

case 0:

break;

case 1:

textBoxParam1.Visible = true;

paramLabel1.Visible = true;

paramLabel1.Text = paramSP[0].ItemArray[1].ToString() + " (" + paramSP[0].ItemArray[2].ToString() + "):";

paramsCount++;

break;

case 2:

textBoxParam1.Visible = true;

paramLabel1.Visible = true;

paramLabel1.Text = paramSP[0].ItemArray[1].ToString() + " (" + paramSP[0].ItemArray[2].ToString() + "):";

paramsCount++;

textBoxParam2.Visible = true;

paramLabel2.Visible = true;

paramLabel2.Text = paramSP[1].ItemArray[1].ToString() + " (" + paramSP[1].ItemArray[2].ToString() + "):";

paramsCount++;

break;

case 3:

textBoxParam1.Visible = true;

paramLabel1.Visible = true;

paramLabel1.Text = paramSP[0].ItemArray[1].ToString() + " (" + paramSP[0].ItemArray[2].ToString() + "):";

paramsCount++;

textBoxParam2.Visible = true;

paramLabel2.Visible = true;

paramLabel2.Text = paramSP[1].ItemArray[1].ToString() + " (" + paramSP[1].ItemArray[2].ToString() + "):";

paramsCount++;

textBoxParam3.Visible = true;

paramLabel3.Visible = true;

paramLabel3.Text = paramSP[2].ItemArray[1].ToString() + " (" + paramSP[2].ItemArray[2].ToString() + "):";

paramsCount++;

break;

default:

break;

}

}

private string GetReportSP() => SqlHelper.GetReportSP(currentReport);

private DataSet GetReportParams() => SqlHelper.GetSPParams(GetReportSP());

/// <summary>

/// CREATE REPORT

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button1\_Click(object sender, EventArgs e)

{

try

{

SqlParameter param1 = new SqlParameter();

SqlParameter param2 = new SqlParameter();

SqlParameter param3 = new SqlParameter();

var ds = new DataSet();

switch (paramsCount)

{

case 0:

break;

case 1:

param1 = CreateSqlParameter(paramLabel1.Text, string.IsNullOrEmpty(textBoxParam1.Text) ? null : textBoxParam1.Text);

ds = SqlHelper.ExecSpWithParams(GetReportSP(), param1);

break;

case 2:

param1 = CreateSqlParameter(paramLabel1.Text, string.IsNullOrEmpty(textBoxParam1.Text) ? null : textBoxParam1.Text);

param2 = CreateSqlParameter(paramLabel2.Text, string.IsNullOrEmpty(textBoxParam2.Text) ? null : textBoxParam2.Text);

ds = SqlHelper.ExecSpWithParams(GetReportSP(), param1, param2);

break;

case 3:

param1 = CreateSqlParameter(paramLabel1.Text, string.IsNullOrEmpty(textBoxParam1.Text) ? null : textBoxParam1.Text);

param2 = CreateSqlParameter(paramLabel2.Text, string.IsNullOrEmpty(textBoxParam2.Text) ? null : textBoxParam2.Text);

param3 = CreateSqlParameter(paramLabel3.Text, string.IsNullOrEmpty(textBoxParam3.Text) ? null : textBoxParam3.Text);

ds = SqlHelper.ExecSpWithParams(GetReportSP(), param1, param2, param3);

break;

default:

break;

}

if (ds != default)

{

dataGridView1.DataSource = null;

dataGridView1.AutoGenerateColumns = true;

dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

dataGridView1.DataSource = ds;

dataGridView1.DataMember = ds.Tables?[0].TableName;

dataGridView1.ReadOnly = true;

}

}

catch(Exception ex)

{

app.LogError(ex);

}

}

private SqlParameter CreateSqlParameter(string param, object value)

{

SqlParameter parameter = new SqlParameter();

var splitParam = param.Split(' ');

parameter.ParameterName = splitParam[0];

parameter.Value = value;

if (splitParam[1].Contains("DATE"))

parameter.SqlDbType = SqlDbType.Date;

else if (splitParam[1].Contains("SMALLINT"))

parameter.SqlDbType = SqlDbType.SmallInt;

else if (splitParam[1].Contains("INT"))

parameter.SqlDbType = SqlDbType.Int;

else if (splitParam[1].Contains("NVARCHAR"))

parameter.SqlDbType = SqlDbType.NVarChar;

return parameter;

}

/// <summary>

/// EXPORT TO PDF

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void exportButton\_Click(object sender, EventArgs e)

{

if (dataGridView1.Rows.Count > 0)

{

SaveFileDialog sfd = new SaveFileDialog();

sfd.Filter = "PDF (\*.pdf)|\*.pdf";

sfd.FileName = $"{currentReport} - {DateTime.Now.ToShortDateString()}.pdf";

bool fileError = false;

if (sfd.ShowDialog() == DialogResult.OK)

{

if (File.Exists(sfd.FileName))

{

try

{

File.Delete(sfd.FileName);

}

catch (IOException ex)

{

fileError = true;

app.LogError(ex);

}

}

if (!fileError)

{

try

{

iTextSharp.text.Image logo = iTextSharp.text.Image.GetInstance(new FileStream(Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location), "../../Resources/logoImage.bmp"), FileMode.Open));

logo.SetAbsolutePosition((PageSize.A4.Width - logo.ScaledWidth), (PageSize.A4.Height - logo.ScaledHeight));

PdfPTable pdfTable = new PdfPTable(dataGridView1.Columns.Count);

pdfTable.DefaultCell.Padding = 3;

pdfTable.WidthPercentage = 100;

pdfTable.HorizontalAlignment = Element.ALIGN\_LEFT;

foreach (DataGridViewColumn column in dataGridView1.Columns)

{

PdfPCell cell = new PdfPCell(new Phrase(column.HeaderText));

pdfTable.AddCell(cell);

}

foreach (DataGridViewRow row in dataGridView1.Rows)

{

foreach (DataGridViewCell cell in row.Cells)

{

pdfTable.AddCell(cell.Value.ToString());

}

}

using (FileStream stream = new FileStream(sfd.FileName, FileMode.Create))

{

Document pdfDoc = new Document(PageSize.A4, 10f, 20f, 20f, 10f);

PdfWriter.GetInstance(pdfDoc, stream);

pdfDoc.Open();

pdfDoc.Add(new Paragraph($"{currentReport} - report, generated {DateTime.Now}.\n\nAuto-reporting system.\n\n"));

pdfDoc.Add(pdfTable);

pdfDoc.Add(logo);

pdfDoc.Close();

stream.Close();

}

app.LogSuccess("Data Exported Successfully!");

}

catch (Exception ex)

{

app.LogError(ex);

}

}

}

}

else

{

app.LogInfo("No records to export!");

}

}

}

}

Класс SqlHelper

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp1.Classes

{

public static class SqlHelper

{

public static string connString = ConfigurationManager.ConnectionStrings["connString"].ConnectionString;

public static IList<string> GetTables(string queryString)

{

var tables = new List<string>();

using (SqlConnection connection = new SqlConnection(connString))

{

SqlCommand command = new SqlCommand(queryString, connection);

connection.Open();

var reader = command.ExecuteReader();

try

{

while (reader.Read())

{

//if(!reader["TABLE\_NAME"].ToString().Contains("Report"))

tables.Add(reader["TABLE\_NAME"].ToString());

}

}

finally

{

reader.Close();

}

}

return tables;

}

public static IList<string> GetReports(string queryString)

{

var tables = new List<string>();

using (SqlConnection connection = new SqlConnection(connString))

{

SqlCommand command = new SqlCommand(queryString, connection);

connection.Open();

var reader = command.ExecuteReader();

try

{

while (reader.Read())

{

tables.Add(reader["ReportName"].ToString());

}

}

finally

{

reader.Close();

}

}

return tables;

}

public static IList<string> GetViews(string queryString)

{

var tables = new List<string>();

using (SqlConnection connection = new SqlConnection(connString))

{

SqlCommand command = new SqlCommand(queryString, connection);

connection.Open();

var reader = command.ExecuteReader();

try

{

while (reader.Read())

{

tables.Add(reader["Name"].ToString());

}

}

finally

{

reader.Close();

}

}

return tables;

}

public static string GetReportSP(string reportName)

{

string reportSP = default;

using (SqlConnection connection = new SqlConnection(connString))

{

SqlCommand command = new SqlCommand(SqlScripts.SelectReportSP + "'" + reportName + "'", connection);

connection.Open();

var reader = command.ExecuteReader();

try

{

while (reader.Read())

{

reportSP = reader["StoredProcedure"].ToString();

}

}

finally

{

reader.Close();

}

}

return reportSP;

}

public static DataSet GetTableDataSet(string name)

{

DataSet ds = new DataSet();

using (SqlConnection connection = new SqlConnection(connString))

{

connection.Open();

SqlDataAdapter da = new SqlDataAdapter();

SqlCommand cmd = connection.CreateCommand();

cmd.CommandText = SqlScripts.SelectScript + GetTableName(name);

da.SelectCommand = cmd;

da.Fill(ds);

}

return ds;

}

public static void SaveAndCommitToDb(DataSet dataset, string currentTable)

{

using (SqlConnection conn = new SqlConnection(connString))

{

conn.Open();

DataSet newDataSet = new DataSet();

SqlDataAdapter newDataAdapter = new SqlDataAdapter();

newDataAdapter.SelectCommand = new SqlCommand(SqlScripts.SelectScript + GetTableName(currentTable), conn);

SqlCommandBuilder cb = new SqlCommandBuilder(newDataAdapter);

newDataAdapter.Fill(newDataSet);

newDataAdapter.UpdateCommand = cb.GetUpdateCommand();

newDataAdapter.Update(dataset, dataset.Tables[0].TableName);

conn.Close();

}

}

public static DataSet GetSPParams(string spName)

{

DataSet ds = new DataSet();

using (SqlConnection connection = new SqlConnection(connString))

{

connection.Open();

SqlDataAdapter da = new SqlDataAdapter();

SqlCommand cmd = connection.CreateCommand();

cmd.CommandText = SqlScripts.SelectSPParams + spName.Split('.')[1] + "'";

da.SelectCommand = cmd;

da.Fill(ds);

}

return ds;

}

public static DataSet ExecSpWithParams(string spName, params SqlParameter[] sqlParameters)

{

DataSet ds = new DataSet();

using (SqlConnection connection = new SqlConnection(connString))

{

connection.Open();

SqlDataAdapter da = new SqlDataAdapter();

SqlCommand cmd = new SqlCommand(spName, connection);

cmd.CommandType = CommandType.StoredProcedure;

foreach (var param in sqlParameters)

cmd.Parameters.Add(param);

da.SelectCommand = cmd;

da.Fill(ds);

}

return ds;

}

private static string GetTableName(string nameWithSchema)

{

var splittedString = nameWithSchema.Split('.');

StringBuilder builder = new StringBuilder();

builder.Append("[");

builder.Append(splittedString[0]);

builder.Append("].");

builder.Append("[");

builder.Append(splittedString[1]);

builder.Append("]");

return builder.ToString();

}

}

}