Projekt Bazy Danych
Temat: Aplikacja wykorzystująca technologię baz danych
Autorzy: Marcin Sznura & Michał Stancelewski

1. Charakterystyka projektu

- **Opis:**

Celem projektu było stworzenie aplikacji bazodanowej "Webflex", symulującej korzystanie z biblioteki filmowej bez wychodzenia z domu. Użytkownicy rejestrują się w systemie i logują na swoje konto, gdzie mogą dodawać środki na konto. Po zaisileniu konta, mają możliwość dodania filmu na stałe do swojej biblioteki za ustaloną opłatą. Informację o filmach, użytkownikach i ich dane przechowywane są w bazie danych. Użytkownik przeprowadza interakcję z bazą za pomocą aplikacji napisanej w języki C#. Sama aplikacja natomiast w interakcji z bazą korzysta ze stworzonych w bazie procedur.

- **Wykorzystane technologie:**

SQL Server:

Do stworzenia bazy danych wykorzystaliśmy MS SQL Server. Zalety tego oprogramowania, które skłoniły nas do jego wyboru to:

- łatwość instalacji

- brak opłat za korzystanie

- odporność na błęd, łatwe tworzenie kopi zapasowych i ich przywracanie

- łatwość integracji z innymi technologiami Microsoft tj. używanie przez nas Visual Studio

Visual Studio:

Aplikację obsługującą naszą bazę danych postanowiliśmy napisać w programie Visual Studio w języku C#. Powodami tego wyboru były:

- C# i .NET może pracować z większością baz danych

- dobrze działa w środowisku Windows

- doskonale współgra z MS SQL Server jako że oba produkty należą do Microsoft

- C# jest językiem kompilowalnym, co zwiększa bezpieczeństwo aplikacji

- **Architektura oprogramowania:**

Aplikacja klienta:

Aplikacja klienta umożliwia użytkownikowi na interakcję z bazą danych. Pozwala na rejestrację w bazie, logowanie za pomocą utworzonego loginu i hasła, dodanie środków na konto, zakup filmów, przeglądanie i filtrowanie ich w bibliotece, zmianę danych oraz usunięcie z bazy danych.

Baza danych:

Baza danych zawiera wszystkie informację o użytkownikach, filmach oraz zbiór procedur, za pomocą których aplikacja klienta wprowadza zmiany i dodaję nowe dane do bazy danych.

2. Baza Danych

• Skrypty tworzące strukturę SZBD

Zarządrzanie naszą bazą danych odbywa się głównie za pomocą 3 skryptów:

- **AddNewUser** - procedura ta dopisuje dane użytkownika do tabeli Users, tworzy oraz wypełnia nową tabelę *userName*_Library przechowującą informację o zakupionych przez użytkowanika filmach oraz doaje widoki i triggery, z których korzysta użytkownik.

```
PROCEDURE [dbo].[AddNewUser]
	@id int ,
	@login nvarchar(20),
	@password nvarchar(20),
	@name nvarchar(20),
	@surname nvarchar(20),
	@mail nvarchar(50),
	@balance money
AS
BEGIN
	SET NOCOUNT ON;
```

```
     INSERT INTO Users (ID, login,password,name,surname,[e-mail],balance)
        VALUES(@id,@login, @password, @name, @surname, @mail, @balance);

       declare @tablename nvarchar(50)
       set @tablename = @login + '_Library'

       DECLARE @UserLibrary varchar(1000)
       SET @UserLibrary = 'CREATE TABLE '+ @tablename+'
       (id int NOT NULL,title varchar(255) NOT NULL,
       bought bit NOT NULL CONSTRAINT '+@login+'_pk PRIMARY KEY(id)) '

       EXEC(@UserLibrary)

       Declare @i int, @j varchar(5)
       Set @i = 1
       Set @j = 1
       DECLARE @PopulateUserLibrary varchar(1000)


       While @i <= 10
       Begin
       SET @PopulateUserLibrary ='INSERT INTO ' + @login + '_Library VALUES ('+@j+',0,0)'
       EXEC(@PopulateUserLibrary)
       Set @j = @j+1
       Set @i = @i + 1
       End


       DECLARE @UserLibrary2 varchar(1000)
       SET @UserLibrary2 ='UPDATE ' + @login + '_Library SET ' + @login + '_Library.id = Movies.id,
' + @login + '_Library.title = Movies.title,
       ' + @login + '_Library.bought = 0 FROM dbo.Movies WHERE ' + @login + '_Library.id =
dbo.Movies.id;'

       EXEC(@UserLibrary2)

    DECLARE @UserView varchar(1000)
       SET @UserView =    'CREATE VIEW '+@login+'_LibraryView AS
       select * from '+@login+'_Library;'

       EXEC(@UserView)


              DECLARE @UserStatTrigger varchar(1000)
       SET @UserStatTrigger = 'CREATE TRIGGER '+@login+'_MovieStats ON '+@login+'_Library
       AFTER UPDATE AS
       BEGIN
       UPDATE MoviesStats SET MoviesStats.sold += 1
FROM   MoviesStats
       JOIN inserted
         ON MoviesStats.movieID = inserted.id
END'
```

```
EXEC(@UserStatTrigger)

END
```

- **BuyingMovieTransaction** - procedura dodające filmy do biblioteki użytkownika oraz pomniejszająca jego saldo o określoną kwotę. Umożliwia podstawową akcję jaką jest nabywanie filmów przez użytkownika.

```
PROCEDURE [dbo].[BuyingMovieTransaction]

	@login varchar(50),
	@cost money,
	@userID int,
	@movieID varchar(5)
AS
BEGIN

	SET NOCOUNT ON;

	BEGIN TRANSACTION
	UPDATE Users
	SET balance = balance - @cost
	WHERE id = @userID;

	DECLARE @Addmovie varchar(1000)
	SET @Addmovie = 'UPDATE '+@login+'_Library
	SET bought = 1
	WHERE id = '+@movieID+';'

	EXEC(@Addmovie)
	COMMIT TRANSACTION

END
```

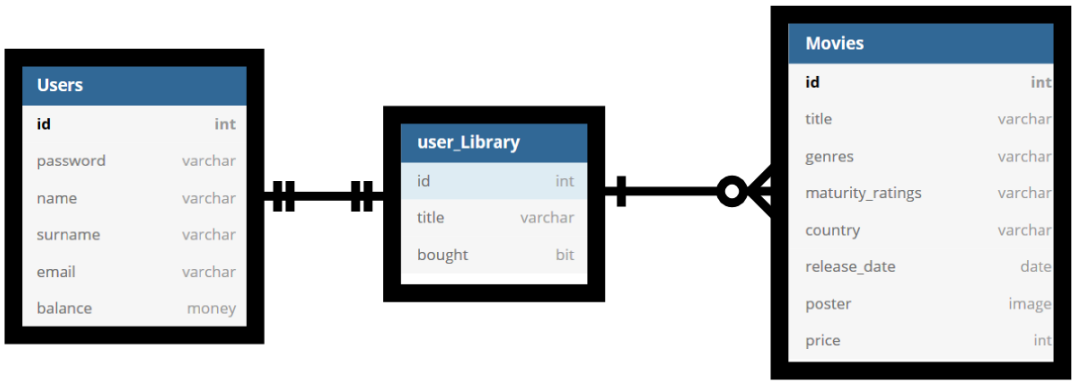
- **DeleteUser** - procedura usuwa dane użytkownika z tabeli Users tabelę będącą biblioteką użytkownika oraz czyści widoki użytkownika.

```
PROCEDURE [dbo].[DeleteUser]
	-- Add the parameters for the stored procedure here
	@login varchar(30),
	@id varchar(5)
AS
BEGIN
	SET NOCOUNT ON;

    BEGIN TRANSACTION

	DELETE FROM Users
	WHERE id = @id;
```

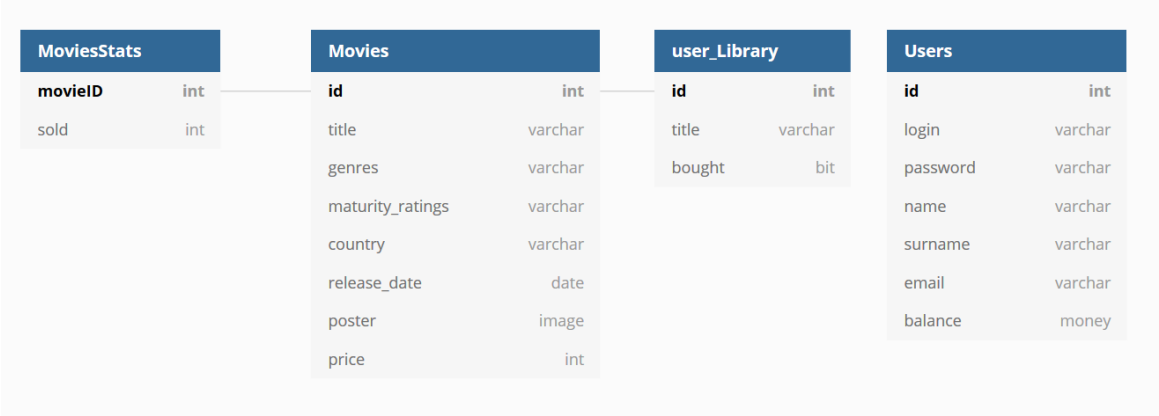
```
        DECLARE @DropUserTable varchar(1000)
        SET @DropUserTable = 'DROP TABLE '+@login+'_Library'
        EXEC(@DropUserTable)

        DECLARE @DropUserView varchar(1000)
        SET @DropUserTable = 'DROP VIEW '+@login+'_LibraryView'
        EXEC(@DropUserTable)

        COMMIT TRANSACTION
END
```

- **Diagramy ER (najciekawsze rozwiązania)**

Każdy użytkownik może musi mieć jedną bibliotekę i każda biblioteka musi mieć użytkownika. W bibliotece użytkownika może znajdować się wiele filmów, ale dany film występuje w bibliotece użytkownika tylko raz.



- **Diagram BD**



3. Przykłady rozwiązań wykorzystujące:

- **Triggery**

Trigger wyślij e-mail użytkownikowi: trigger podczas rejestracji symuluje wysyłanie e-maila potwierdzającego rejestrację, poprzez utworzenie dokumentu tekstowego na dysku.

```
TRIGGER [dbo].[signUpEmail]
ON [dbo].[Users]
AFTER INSERT
AS
BEGIN
DECLARE @query NVARCHAR(2000)
DECLARE @outText NVARCHAR(2000)
SELECT @outText = 'Hi '+ RTRIM(name) +'! Thank you for joining Weblex. You can sign up using your
login: '+ RTRIM(login)+' and your password.'
FROM inserted
SET @query = 'master..xp_cmdshell ''echo ' + @outText + ' > E:\IO\email.txt'' '
EXEC ( @query )
END
```

Trigger zmień statystyki filmu: trigger podczas sprzedaży filmu zwiększa statystyki jego sprzedaży w tabeli MoviesStats.

```
TRIGGER [dbo].[movieStats_q]
ON [dbo].[q_Library]
AFTER UPDATE
AS
BEGIN
UPDATE MoviesStats
SET    MoviesStats.sold += 1
FROM   MoviesStats
       JOIN inserted
         ON MoviesStats.movieID = inserted.id
END
```

- **Widoki**

Widok wszystkich filmów w tabeli Movies:

```
SELECT id, title, genres, [maturity ratings], country, [release date], poster, price FROM dbo.Movies
```

Widok filmów z podziałem na gatunek:

```
SELECT id, title, genres, [maturity ratings], country, [release date], poster, price FROM dbo.Movies
WHERE (genres = 'GATUNEK FILMU')
```

Widok danych użytkownika:

```
SELECT id, login, password, name, surname, balance, [e-mail]
FROM dbo.Users
```

Widok filmów z podziałem na grupy wiekowe:

```
SELECT id, title, genres, [maturity ratings], price, poster, [release date], country FROM dbo.Movies
WHERE ([maturity ratings] = N'GRUPA WIEKOWA')
```

Widok biblioteki użytkownika:

```
SELECT        id, title, bought
FROM            dbo.q_Library
```

- **Procedury**

Procedura dodające nowego użytkownika: dopisuje dane użytkownika do tabeli Users, tworzy tabelę *user*_Library i wypełnia ją danymi filmów, tworzy widok biblioteki użytkownika oraz trigger aktualizujący ilość sprzedanych filmów:

```
PROCEDURE [dbo].[AddNewUser]
       @id int ,
       @login nvarchar(20),
       @password nvarchar(20),
       @name nvarchar(20),
       @surname nvarchar(20),
       @mail nvarchar(50),
       @balance money
AS
BEGIN
       SET NOCOUNT ON;

     INSERT INTO Users (ID, login,password,name,surname,[e-mail],balance)
        VALUES(@id,@login, @password, @name, @surname, @mail, @balance);

       declare @tablename nvarchar(50)
       set @tablename = @login + '_Library'

       DECLARE @UserLibrary varchar(1000)
       SET @UserLibrary = 'CREATE TABLE '+ @tablename+'
       (id int NOT NULL,title varchar(255) NOT NULL,
```

```
        bought bit NOT NULL CONSTRAINT '+@login+'_pk PRIMARY KEY(id)) '

        EXEC(@UserLibrary)

        Declare @i int, @j varchar(5)
        Set @i = 1
        Set @j = 1
        DECLARE @PopulateUserLibrary varchar(1000)


        While @i <= 10
        Begin
        SET @PopulateUserLibrary ='INSERT INTO ' + @login + '_Library VALUES ('+@j+',0,0)'
        EXEC(@PopulateUserLibrary)
        Set @j = @j+1
        Set @i = @i + 1
        End


        DECLARE @UserLibrary2 varchar(1000)
        SET @UserLibrary2 ='UPDATE ' + @login + '_Library SET ' + @login + '_Library.id = Movies.id,
' + @login + '_Library.title = Movies.title,
        ' + @login + '_Library.bought = 0 FROM dbo.Movies WHERE ' + @login + '_Library.id =
dbo.Movies.id;'

        EXEC(@UserLibrary2)

    DECLARE @UserView varchar(1000)
        SET @UserView =     'CREATE VIEW '+@login+'_LibraryView AS
        select * from '+@login+'_Library;'

        EXEC(@UserView)


               DECLARE @UserStatTrigger varchar(1000)
        SET @UserStatTrigger = 'CREATE TRIGGER '+@login+'_MovieStats ON '+@login+'_Library
        AFTER UPDATE AS
        BEGIN
        UPDATE MoviesStats SET MoviesStats.sold += 1
FROM    MoviesStats
        JOIN inserted
          ON MoviesStats.movieID = inserted.id
END'

EXEC(@UserStatTrigger)

END
```

Procedura zakupu filmu: odejmuje koszt filmu od salda użytkownika i dopisuje film do jego biblioteki:

```
PROCEDURE [dbo].[BuyingMovieTransaction]

	@login varchar(50),
	@cost money,
	@userID int,
	@movieID varchar(5)
AS
BEGIN

	SET NOCOUNT ON;

	BEGIN TRANSACTION
	UPDATE Users
	SET balance = balance - @cost
	WHERE id = @userID;

	DECLARE @Addmovie varchar(1000)
	SET @Addmovie = 'UPDATE '+@login+'_Library
	SET bought = 1
	WHERE id = '+@movieID+';'

	EXEC(@Addmovie)
	COMMIT TRANSACTION

END
```

Procedura usuń użytkownika: usuwa dane użytkownika z tabeli Users, tabelę biblioteka użytkownika oraz widok tejże tabeli:

```
PROCEDURE [dbo].[DeleteUser]
	@login varchar(30),
	@id varchar(5)
AS
BEGIN
	SET NOCOUNT ON;

    BEGIN TRANSACTION

	DELETE FROM Users
	WHERE id = @id;

	DECLARE @DropUserTable varchar(1000)
	SET @DropUserTable = 'DROP TABLE '+@login+'_Library'
	EXEC(@DropUserTable)

	DECLARE @DropUserView varchar(1000)
	SET @DropUserTable = 'DROP VIEW '+@login+'_LibraryView'
	EXEC(@DropUserTable)
```

```
        COMMIT TRANSACTION
END
```

Procedura filtrująca bibliotekę użytkownika: wyświetla tylko kupione filmy z danego gatunku:

```
PROCEDURE [dbo].[FilterLibrary]
        @login varchar(30),
        @genres varchar(30)
AS
BEGIN
        SET NOCOUNT ON;

        DECLARE @LibraryFilter varchar(1000)
        SET @LibraryFilter = 'SELECT dbo.Movies.id, dbo.Movies.title, dbo.Movies.genres,
dbo.Movies.[maturity ratings], dbo.Movies.country, dbo.Movies.[release date], dbo.Movies.poster,
dbo.Movies.price, dbo.' + @login + '_Library.id AS Expr1,
                                dbo.' + @login + '_Library.bought
        FROM dbo.Movies INNER JOIN
                                dbo.' + @login + '_Library ON dbo.Movies.id = dbo.' + @login +
'_Library.id
        WHERE(dbo.Movies.genres = '''+@genres+''') AND(dbo.' + @login + '_Library.bought = 1)'

           EXEC(@LibraryFilter)
END
```

Procedura filtrująca sklep użytkownika: wyświetla tylko niekupione filmy z danego gatunku:

```
PROCEDURE [dbo].[FilterShop]
        @genres varchar(20),
        @login varchar(30)
AS
BEGIN
        SET NOCOUNT ON;

    DECLARE @ShopFilter varchar(1000)
        SET @ShopFilter = 'SELECT dbo.Movies.id, dbo.Movies.title, dbo.Movies.genres,
dbo.Movies.[maturity ratings], dbo.Movies.country, dbo.Movies.[release date], dbo.Movies.poster,
dbo.Movies.price, dbo.' + @login + '_Library.id AS Expr1,
                                dbo.' + @login + '_Library.bought
        FROM dbo.Movies INNER JOIN
                                dbo.' + @login + '_Library ON dbo.Movies.id = dbo.' + @login +
'_Library.id
        WHERE(dbo.Movies.genres = '''+@genres+''') AND(dbo.' + @login + '_Library.bought = 0)'

            EXEC(@ShopFilter)

END
```

- **Transakcje**

Transakcja kupna filmu: Obie akcje - odjęcie środków oraz dodanie filmu do biblioteki użytkownika muszą zostać wykonane.

```
BEGIN TRANSACTION
UPDATE Users
SET balance = balance - @cost
WHERE id = @userID;

DECLARE @Addmovie varchar(1000)
SET @Addmovie = 'UPDATE '+@login+'_Library
SET bought = 1
WHERE id = '+@movieID+';'

EXEC(@Addmovie)
COMMIT TRANSACTION
```

Procedura usuń użytkownika: Wszystkie dane użytkownika muszą zostać usunięte. Jego informacje z tabeli Users, biblioteka oraz widok.

```
BEGIN TRANSACTION

    DELETE FROM Users
    WHERE id = @id;

    DECLARE @DropUserTable varchar(1000)
    SET @DropUserTable = 'DROP TABLE '+@login+'_Library'
    EXEC(@DropUserTable)

    DECLARE @DropUserView varchar(1000)
    SET @DropUserTable = 'DROP VIEW '+@login+'_LibraryView'
    EXEC(@DropUserTable)

    COMMIT TRANSACTION
```