

Bachelor's thesis
Information Technology
Specialization: Internet Technology
2013

Aman Yadav

WINDOWS PRESENTATION FOUNDATION APPLICATION DEVELOPMENT FOR A MOVIE THEATER



TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES
Information Technology
08.05.2013 | 49 pages
Instructor: Balsam Almurrani

Aman Yadav

Windows Presentation Foundation Application Development for a Movie Theater

This thesis is mainly focused on describing a WPF application for a movie theater developed by using Microsoft Visual Studio, Expression Blend and Microsoft SQL server.

The thesis begins with the description of platforms, relevant technology, and the requirement analysis during the application development processes. Here, MS Visual Studio has been used as the main programming platform, whereas the MS expression Blend is responsible for the design and animation, and finally the MS SQL server is responsible for handling the database.

The rest of the thesis covers the application description, analysis, design, and implementation of the application. It also includes the structuring of the database, testing portion, and handling the exception and errors.

KEYWORDS:

C#, MS SQL server. WPF, LINQ.

CONTENT

1 INTRODUCTION	7
2 PLATFORMS AND RELEVANT TECHNOLOGIES	8
2.1 Visual Studio	8
2.2 MS Expression Blend	8
2.3 MS SQL Server	8
2.4 Linq	8
3 APPLICATION	10
3.1 WPF	10
3.2 Movie Theater Application Functional Scenario	10
3.2.1 User Authentication	10
3.2.2 Data Connection Establishment	10
3.2.3 User Interface Decission	10
3.2.4 Data Transferring and Updating the Database	11
4 REQUIREMENT ANALYSES	12
4.1 System Requirements	12
4.2 Application Requirements	12
4.3 Security Requirements	12
4.4 Functional Requirements	13
5 DATABASE ANALYSIS AND DESIGN	14
5.1 Database Design	14
5.1.1 Relational Model	15
5.1.2 Entity-Relation Model	15
5.1.3 Normalization	16
5.1.4 Referential Integrity	18
5.2 Background Solution	19
6 GRAPHICAL USER INTERFACES DESIGN	20
6.1 Login Window	20
6.2 Admin Window	20
6.3 Manager Window	28
6.4 Cashier Window	31
7 DATABASE MODULES	40
7.1 Tables	40

7.1.1 Property Tables	40
7.1.2 Table Definition	42
7.2 Queries	43
7.2.1 Multi Join, Where	43
7.2.2 Groupby, Having, Virtual Column	43
7.2.3 Except, Intersect And Union Expression	44
8 TESTS AND FIXES	45
8.1 Compatibility Testing	45
8.2 Functional Testing	45
8.2.1 Run-Time Problems	45
8.3 Security Testing	45
9 FUTURE IMPROVEMENT	47
10 CONCLUSIONS	48
REFERENCES	49
FIGURES	
Figure 1. Mapped LINQ diagram.	9
Figure 2. Process of designing a database.	15
Figure 3. Relational Model.	15
Figure 4. Entity-Relation Model before Normalization.	16
Figure 5. Entity-Relation Model after Normalization.	18
Figure 6. Referential Integrity Diagram.	19
Figure 7. Login Window.	20
Figure 8. Starting Administrator Panel.	21
Figure 9. Worker's Table.	21
Figure 11. Add window.	23
Figure 10. Add window, filling rows.	23
Figure 12. Table after updating with new row.	23
Figure 13. Deleting a row.	24
Figure 14. Selecting Row.	25
Figure 15. Modify rows.	25
Figure 16. Rows after modification.	26
Figure 17. Windows with filled fields.	26
Figure 18. Changing window.	27
Figure 19. Manager Window.	28
Figure 20. Additional Project.	29
Figure 21. Using the references and namespace.	30
Figure 22. Start window cashier.	31
Figure 23. Filtered data and displayed in the grid.	33
Figure 24. DatePicker with BlackOutDays	33
Figure 25. Dynamically filling ticket.	34
Figure 26. Window with free and busy seats.	35
Figure 27. Ticket with chosen seats and price.	37
Figure 28. Sold Tickets Data Grid with new added tickets.	39

TABLES

Table 1. Excerpt of code by LINQ.	9
Table 2. Binding updates Grid in C#.	22
Table 3. XAML Animation Code.	22
Table 4. Code to start Animation dynamically.	22
Table 5. C# code, adding rows, constraints.	24
Table 6. C# code of deleting rows.	25
Table 7. C# code fills fields.	26
Table 8. Modifying record C# and saving it into database.	26
Table 9. XAML template code.	27
Table 10. Using template in Combo Box component.	27
Table 11. C# Deleting items from listbox using key.	27
Table 12. Code for creating the path.	28
Table 13. Code to represent dll file.	29
Table 14. Code to show how our Excel chart should look like.	30
Table 15. The code to use references and namespace.	31
Table 16. Updating data grid, binding, linq queries.	32
Table 17. C# code which writes and splits current data (month, day, year).	33
Table 18. C# code which sets blackout dates.	33
Table 19. Filtering data in C#.	34
Table 20. C# code of a dynamically filled ticket.	35
Table 21. Linq, chosing occupied seats.	36
Table 22. Creating grid with seats.	36
Table 23. Clicking the button dynamically changes the seats template.	37
Table 24. Creating a list of selected seats.	38
Table 25. Find rest of attributes to make a LINQ add Ticket query.	38
Table 26. Property Table.	40

ACRONYMS, ABBREVIATIONS AND SYMBOLS

C#	C Sharp
WPF	Windows Presentation Foundation
SQL	Structured Query Language
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
.NET	Network Enable Technology
MS SQL	Microsoft Structured Query Language
LINQ	Language Integrated Query
DBMS	Database Management System
DBLC	Database Life Cycle
ER	Entity Relationship
RID	Referential Integrity Diagram
RD	Referential Diagram
NF	Normal Form

1 INTRODUCTION

This thesis is mainly focused on the development of a C# application for a movie theatre with the help of the tools provided by Microsoft Visual Studio and, Microsoft Expression Blend for the coding and design portion. In the meantime, it uses Microsoft SQL server 2008 R2 in order to store and update the database.[8][9] To put ourselves in a real life situation, cinema management system has been chosen, which turns out to be a good and challenging basis that creates several situations unique for this kind of project. The plan is to create a couple of user interface that would be used in different fields of movie theatre management (like selling the tickets, updating the database and keeping the track tickets sold), and finally it was decided to create three user interfaces which are most crucial: cashier, administrator, and manager. All of them are based on user interactions, so a GUI for each of them has been created, using different logics, based on connections with database. To make the application work in most natural way, the objects and types, suitable for particular problems were created.

Having stated the problems, and getting our database ready, it was a bit hard to decide whether the application is really important from the cinema point of view. Does our application really work fine with the database of the theatre? How useful is it?? Does our design make it easy to use or not? And finally, the decision was made, and it was positive. Those three GUI are crucial for our cinema applications to work. Here, the Cashier window enables us to: (i) sell tickets, (ii) browse through possible shows, and (iii) give the basics of tickets management. The Administrator window poses as an indispensable factor to consider in a database project, because it provides the clients with a whole software package. This application provides the basics of database management, so users do not have to make use of external software for this purpose. Finally, the Manager tool is added, which can be understood as a statistics tool for comparing data according to any given input. This tool has been extended with the ability to prepare Microsoft Excel files with charts by presenting chosen data. After having our all the user interface and database ready it is supposed to have an application ready to work on a cinema theatre database created earlier, and an application that could fit the needs of all the three different types of users.

After all the theoretical assumptions, it was necessary to test the application in a real life situation and try to see whether all the needed features are able to meet the user requirements, and finally to gather the information and feedback from the users. Garnering this kind of information enables us to figure out better plans and approaches, to make the design processes appear more complex, but comprehensible in such a creative and beneficial way during the proceeding stages. At this stage, it was established that there were a few functions that should be implemented to make the programs run better. Some of them were reliable and easy to use in connection with the database, which enables us to obtain various data, refresh them and operate on its modules that are responsible for controlling the input, so that data in database remain integral. Some user features include rows and seats selections that could be done easily with much convenience.

2 PLATFORMS AND RELEVANT TECHNOLOGIES

2.1 Visual Studio

Visual studio .NET is a Microsoft visual programming environment for programming languages and development tools in order to create web services using XML (Extensible Markup Language). The Visual studio comes with .Net framework including common language runtime itself that provides the visual interface for identifying it as a web services, forms for building user interface that can even support the mobile devices, and it also provides the facility for data integration and debugging.[4] During the development of the application for a movie theater Visual Studio has played a vital role, as it is the main programing environment for the coding of an application in C# language.

2.2 MS Expression Blend

Microsoft Expression Blend is a Microsoft user interface design tools developed by Microsoft. This tool is mainly responsible for creating a WPF and Silver light desktop application which uses XAML for design and animation. It also provides a sophisticated environment for the development of an application by adding multimedia files, such as audios and videos, in order to make the user interface more attractive and easy to use. MS Expression blend is the main tool that was used in order to design the application user interfaces and add the animation which was not possible to do with the Visual Studio.

2.3 MS SQL Server

The MS SQL server is a relational database management system of Microsoft written in C++. It is responsible for storing and retrieving the data using an application running on the same computers or on different computers that is connected to an internet. The popularity of MS SQL server has taken over other database servers because of its high rate of performance in a more secure way as it can perform 1 million queries per seconds and can process 14.8 million email messages per second.[7] We used the MS Sql server during our application development because it is easy to setup and can easily create the object for each items in the database using LINQ.

2.4 Linq

LINQ is one part of Microsoft .NET technology which enables making queries using objects. Linq syntax is easy and similar to SQL. LINQ was useful for our applicaiton development as it treats databases and their elements like objects. In order to work properly, LINQ has to know the whole mapping of databases which will return a collection of objects. The collection can be modified and then put back to the source. Here is the mapped LINQ diagram for our application.

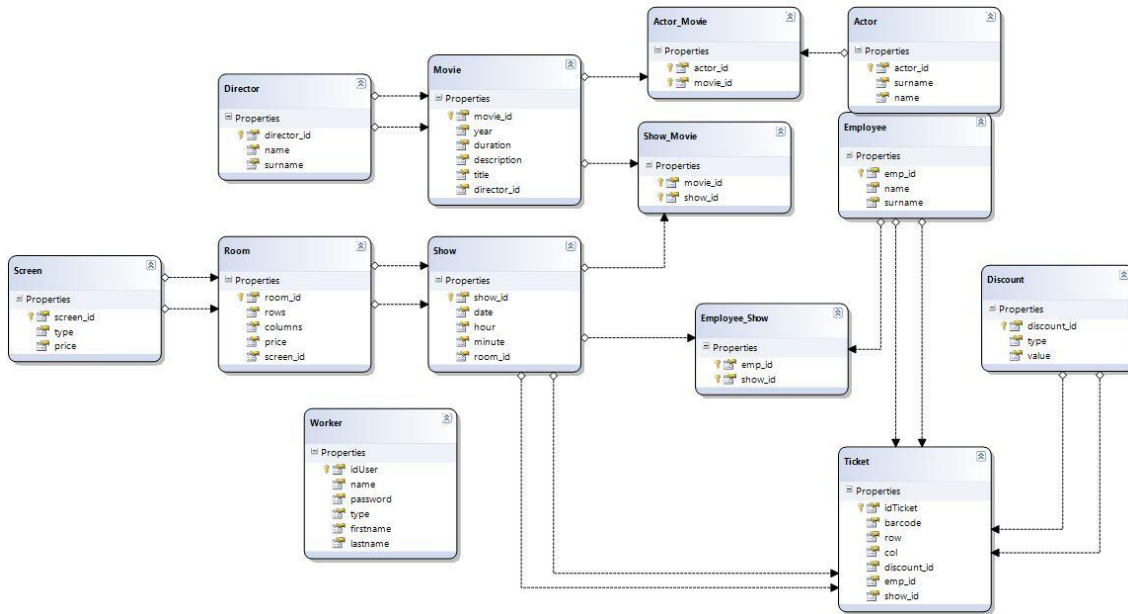


Figure 1. Mapped LINQ diagram.

After generating the database, LINQ draws a diagram as shown above of our database and then it maps our database to objects as shown below in the Excerpt of code by LINQ.

Table 1. Excerpt of code by LINQ.

```

else if (time.Equals("14:00-18:00"))
{
    TicketDataGrid.ItemsSource = null;

    var filteredData = (from show in con.Shows
                        join showMovie in con.Show_Movies
                          on show.show_id equals showMovie.show_id
                        join movie
                          in con.Movies on showMovie.movie_id equals movie.movie_id
                        where show.hour >= 14 && show.hour <= 18 && show.date == date
                        select new { show.show_id, show.date, show.room_id, show.hour, show.minute, movie.title }).ToList();

    TicketDataGrid.ItemsSource = filteredData;
}

```

3 APPLICATION

3.1 WPF

Windows Presentation Foundation (WPF) is a graphical subsystem developed by Microsoft. It is based on work with XAML files. WPF provides the facility of implementing the relationship between design and developers which was slightly harder previously as the design was easier with the tools such as Photoshop but those designs needed strenuous efforts by the developers in order to create the control that could work fine with the designed buttons and the code. In addition, as WPF is based on XAML, it has given the facility for the developers to implement animation and effects such as embedding the audio and video to their application with the help of MS Expression Blend tools, which could make the application much more attractive and impressive to the users.

3.2 Movie Theater Application Functional Scenario

3.2.1 User Authentication

As the application starts it opens the first UI called login window where the users have to input their credentials such as User Name and Password in order to verify their identity on the basis of their status. Here the level of the user, i.e., admin(A), manager(M) or cashier(C) is determined from the username provided and finally the decision is done accordingly. In order to prevent the password being seen from others such as hackers, the password field has been used but in the database it is still stored in the clear string format which is risky and will be dealt with in the future.

3.2.2 Data Connection Establishment

At this point, the connection between the application and database is established with the credentials provided for the MS SQL server which are not the same credentials as entered before. The credentials entered for in the Login window are for the application to decide the level of users and open the user interface accordingly. However, the credentials for the database are the ones which are needed to connect the database with the application and are needed to be entered before the application starts which are the same every time and for every user.

3.2.3 User Interface Decision

At this stage with the credentials provided in the login window, the decision for the user interface to be displayed is made according to the status of the user. There are three user applications for the application which are as follows:

- Cashier Window
- Admin Window
- Manager Window

3.2.4 Data Transferring and Updating the Database

Here, the decision for the UI is made and the required data for the UI is fetched from the server and is displayed in the tables and other respective fields of the UI. Then the modification in the database is done according to the requirements with the help of the respective UI and finally the database is updated in order to keep the track of the sales and purchase of tickets on a particular date and time.

4 REQUIREMENT ANALYSES

The minimum requirements for the application to run smoothly and in an efficient manner are described below.

4.1 System Requirements

The minimum system requirements by an application in order to run are the same as the minimum requirement needed by .NET framework. This is because all the libraries called by the code do not comply with the assembly and these libraries are needed in order to run the application. The requirements are as follows:

- Processor: Minimum Pentium II-450Mhz
- Operating System: Windows 2000 (Server or Professional), Windows XP, or Windows NT 4.0 Server
- Memory: 96 MB for Windows 2000 Professional, 192MB for Windows 2000 server
- Hard drive: 500MB free on the Hard Drive

4.2 Application Requirements

The application requirements for the application are as follows:

- The application should provide the services 24/7.
- It should be fast, simple and easy to use, meaning that it should not jam or crash when the number of users increase. Our application meets these criteria. It is fast in a way because it does not need any installation and takes less than 5 seconds to start. It is simple and easy because the GUI of the application is so clear that the user will not have a problem to use it. In addition it has been designed in a way that mostly it is not necessary to type which makes it simpler and error free.
- If the application crashes due to some reason, then it should be able to regain its state in a short time as the application does not need any installation procedure and can restart in less than 5 seconds.

4.3 Security Requirements

Security requirements for an application are highly important and they include the following measures:

- Only those users with specific privilege should be allowed to use the application.
- Only the admin of the company should be allowed to make modifications in the database except the ticket table as the Cashier can also make modifications on it.
- The application should be protected from being affected by harmful external agents such as virus, hackers, key logger, etc. by having the anti-virus software running in the host PC and not letting any serious commands such as drop and delete to be executed on the database.
- In case the system is affected by an external agent then it should not let it to upload any harmful files to the databases or download anything either by preventing the modification commands on the database such as update, delete,

insert and drop. If the user notices that there is no antivirus in the PC and the application is not functioning properly, then the connection to the database should be terminated manually immediately in order to minimize the risk of damage.

- Access rights for each user:
 - Cashier: The Cashier is responsible for selling the tickets so he or she should just be allowed to make modifications only on the ticket table of the database, that is, either to sell it or cancel it.
 - Manager: The Manager is responsible for keeping the track for the movie tickets sold during the particular period of time for a particular movie and creating a chart for it accordingly. So basically he does not have access to making modifications in the database.
 - Admin: The Admin has the full control in the database as he/she is the one to add or remove a movie in the database.
- The database should be manually backed up once a day or on an alternative day depending on the situation in order to keep the database safe.

4.4 Functional Requirements

Functional requirements basically depend on the type of the user logged in. The requirements are as follows:

- Cashier: The Cashier is responsible for selling the tickets and thus he or she can see the list of movies by any order such as date, actor, release date, etc. and sell the ticket to the customer accordingly. He or she can even cancel the ticket and delete it from the database previously sold.
- Manager: The Manager is responsible for keeping the track of everything going in the business so he can see all the list of movies and the business done by them. From these statistics, he or she can decide which movie is supposed to continue and which one needs to be replaced in the theater in order to enhance the business.
- Admin: The Admin is responsible for adding the new movies, deleting the old ones, adding and removing the users, giving rights to the users and finally backing up the database.

5 DATABASE ANALYSIS AND DESIGN

5.1 Database Design

There are several issues that have to be considered in the design phase of a database, such as identification, analyses and refinement of the business rules, thus making a communication tool, an ER model. The ER model will be used as communication between the user and (database designer) as a blue print for the project.

By designing appropriate data repositories of integrated information, the integrated data must be decomposed properly into its constituent parts, with each part stored in its own table. The table is the basic building block of database design. Consequently, the table's structure is of great importance, since poor table structures lead to redundancy. So we need to prevent poor table structures and produce good table structures. Thereby, we need to consider normalization, which is process for evaluating and correcting a table's structures to minimize data redundancies, and thus reducing the likelihood of destructive data anomalies. Normalizing works through a series of stages called normal forms. For most purposes in business database design, a third normal form is as high as one needs to go in the normalization process, which is also required for this project.

Further, the relationships between the tables must be carefully considered so that the integrated view of the data can be re-created later as information for the end user. The database must facilitate data management and generate accurate and valuable information. The customer may want to generate complex information from the database. Complex information requirements may dictate data transformation, thus they may expand the number of entities and attributes within the design. Therefore, the database may have to sacrifice some of its "clean" design structures and/or some of its high transaction speed to ensure maximum information generation.

We also had to consider end-user demand for fast performance. Therefore, we may be occasionally expected to denormalize some portions of a database design in order to meet performance requirements. However, the price one will pay for increased performance through denormalization is greater data redundancy.

Furthermore, we have to choose which DBMS (Database Management System) to use in this project. There are many to choose from, since this project will be a product of corroboration between two subjects, we choose to use Microsoft SQL Server, because the front end of the application will be developed with Microsoft technology, which will thus enable better integration.

For the back-end of the Cinema application we choose to use the database life cycle(DBLC), which is a never ending life cycle as the monitoring, modification and maintenance life cycle keeps rotating once the database has been implemented. However, at this point we have used a simpler version of the DBLC.

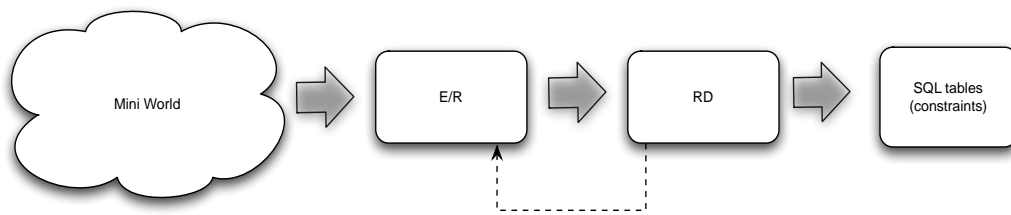


Figure 2. Process of designing a database.

The above diagram shows the process of designing a database in this thesis. An entity-relationship (ER) diagram was created based on assessments of our mini world. In the RD phase, normalization was applied to correct any problems associated with poor design (i.e., redundancy prevention) where it is necessary, and at last in the final phase where the database is implemented. Furthermore, the Chain model was used for the ER diagram, since the ER model is a communication tool between the designer and customer, it would be easier for the customer to understand.

5.1.1 Relational Model

The term relational database was originally defined by Edgar Codd at IBM Almaden Research Center in 1970.[2] It was based on predicate logic and set theory.

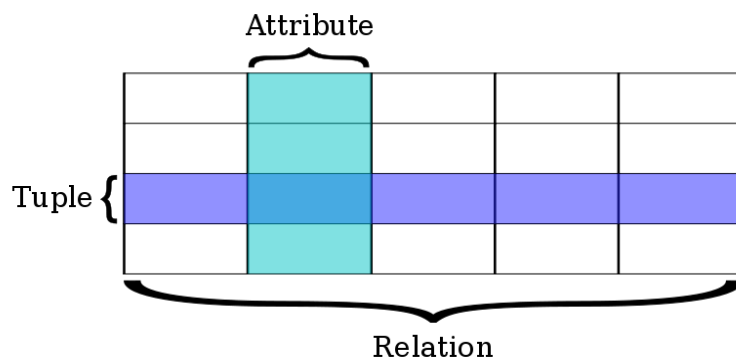


Figure 3. Relational Model.

The main construct for representing data in the relational model is a relation which is defined as a set of tuples that have the same attributes. A tuple represents an object and information about that object.

Although there are other models such as the hierarchical and network model, the relational database is used for this project because it allows the definition of data structure, storage, retrieval operations and integrity constraints.

5.1.2 Entity-Relation Model

Data modelling is the first step in the database design, serving as a bridge between the real-world object and the database model that is implemented. Therefore, the importance of the data-modelling details, expressed graphically through an entity relationships (ER) diagram is essential.

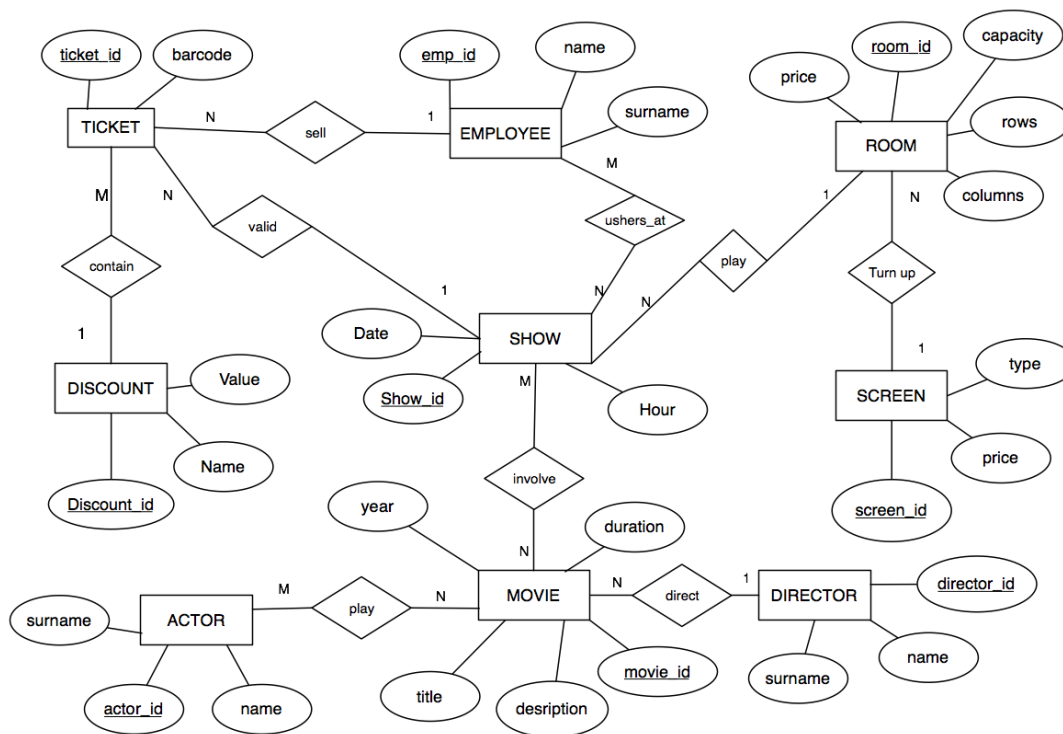


Figure 4. Entity-Relation Model before Normalization.

The ER diagram above represents the conceptual database as viewed by the end user. It reflects the problem description through a database's main components: entities, which represent a real world, objects such as a movie and their characteristics through attributes and the association between the entities as relationships.

Further, the entity-relationship diagram was mapped to table schemes where entities correspond to tables and attributes as table rows. The one-to-many relationships were translated as follows:

- Add the primary key attribute (or attributes) of the entity on the one side of the relationship as a foreign key in the relation on the right side
- The one side migrates to the many side.

The many-to-many relationship:

- Create another relation and include primary keys of all relations as primary key of new relation in RID

The following three tables were created as a bridge between the many-to-many relationships:

- Employee_Show (between EMPLOYEE and SHOW).
- Show_Movie (between SHOW and MOVIE).
- Movie_Actor (Between MOVIE and ACTOR).

5.1.3 Normalization

The normalization process is a very efficient method to prevent redundancies, thereby avoiding data anomalies in the database. After the initial design had been complete, normalization was used to analyse the relationships that exist among the attributes within each entity, to determine if the structure can be improved through normalization. Furthermore, there were no changes made to the initial design, since it met the conditions of 1NF. As for the 1NF, the table structures were checked for not repeating

groups and all primary keys were identified. As for the 2NF, the tables must meet the condition of 1NF and no partial dependencies. Finally, the 3NF must meet the conditions required by 1NF, 2NF and no transitive dependencies. In our case before normalization, we noticed some mistakes in our database. In addition, before normalization, we noticed a redundancy in our database (for example, before we had attribute “capacity” in table Room, which could be calculated by multiplying the attribute row with the column of Room table). Moreover, the attribute (name and surname) was not divided into two another attributes in the Actor table. The next problem was the table screen – we did not have it before. We had the field ‘screen’ in the table Room, but we realised that screen do not have to be only in this table, but it could show on another one in the future, so we added the table Screen. Thus, the bridge tables for many-to-many relationships between the tables such as Employee –Show, Show-Movie and Movie-Actor were created. Finally, we achieved the goal of 3NF in our database.

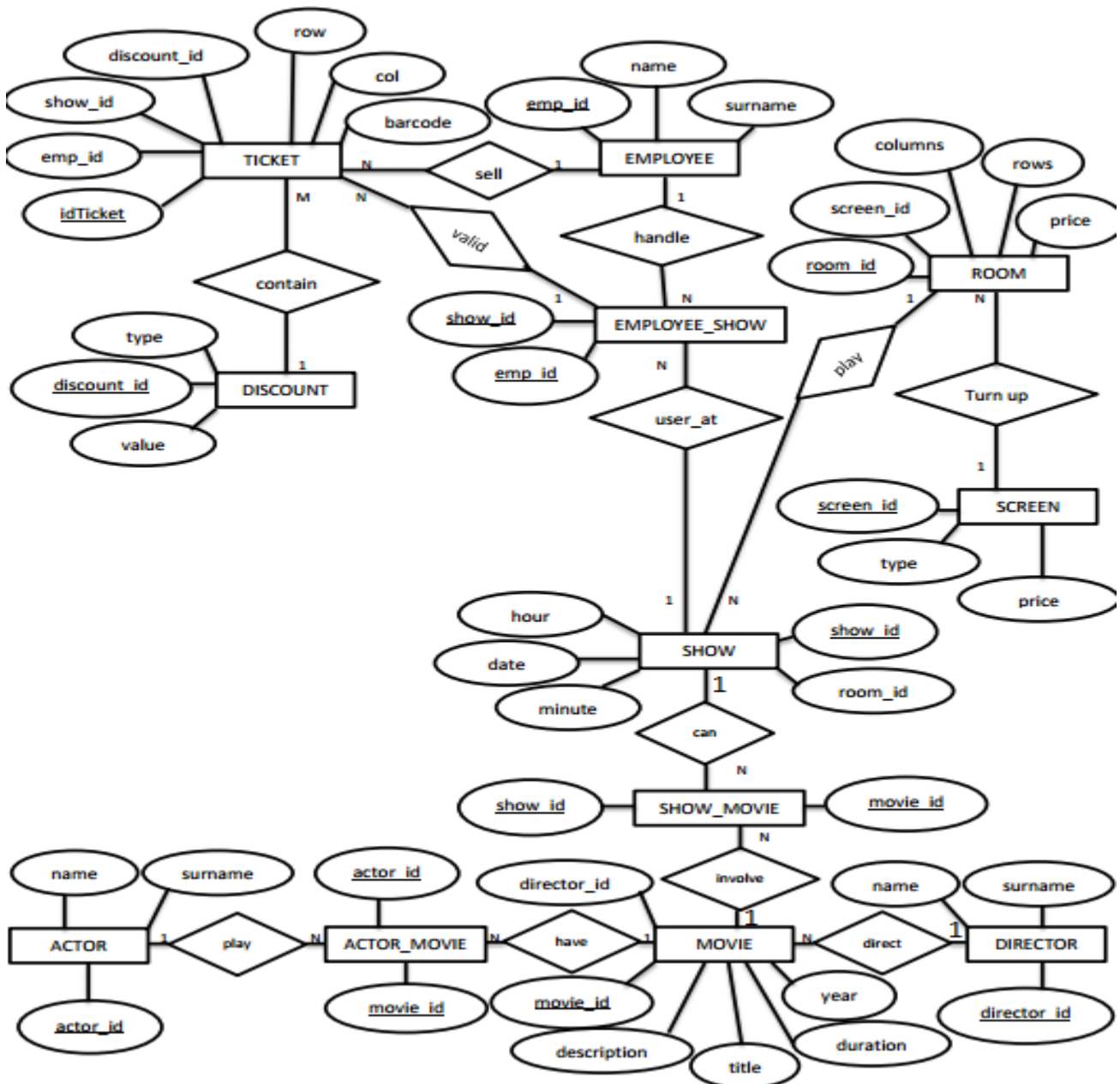


Figure 5. Entity-Relation Model after Normalization.

5.1.4 Referential Integrity

The following referential integrity diagram (RID) shows the tables structure that was mapped from the ER diagram.

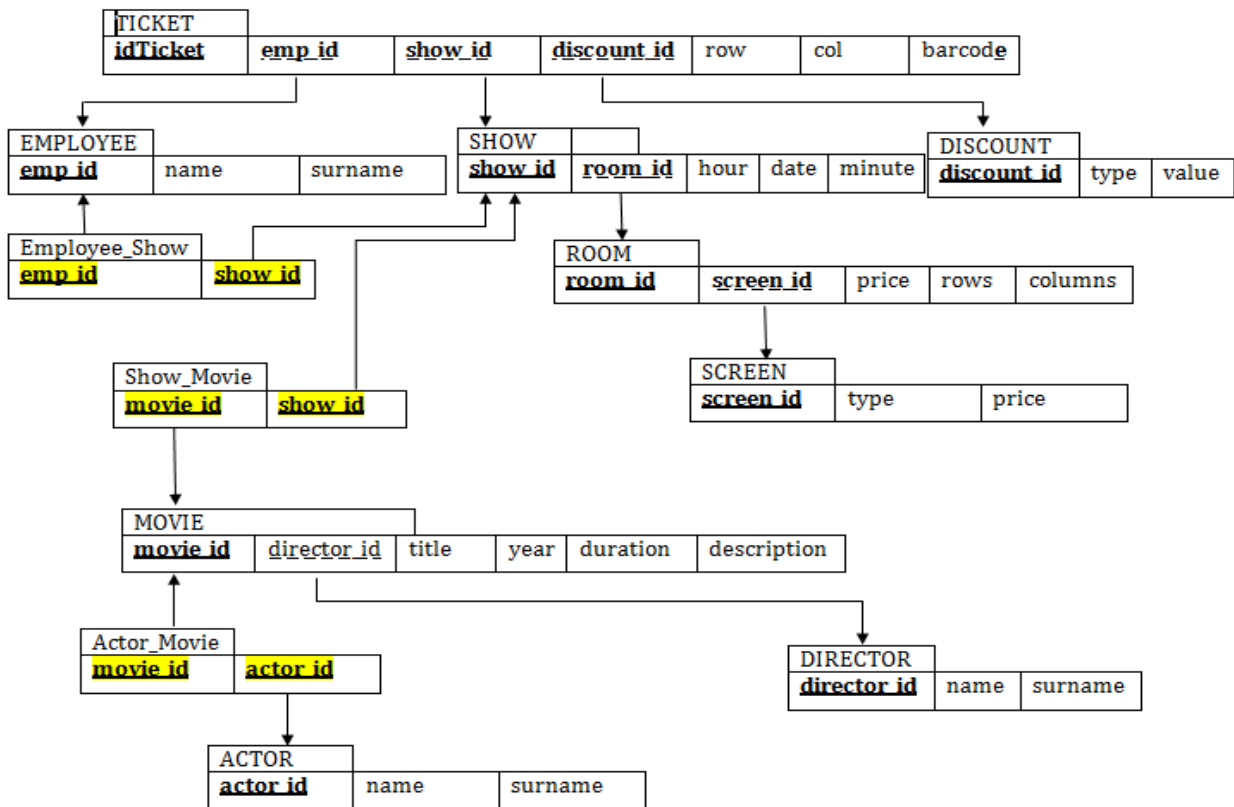


Figure 6. Referential Integrity Diagram.

The RID was made to show the relationships between primary and foreign keys in the tables structure of the database. The RID will illustrate that there are no referential integrity violation between the foreign key candidate key and the primary key, as seen in the above RID, which is a good indication that the foreign key candidate can be accepted as the foreign key. This ensures that relationships between tables remain consistent.

5.2 Background Solution

- Data integrity – wherever our program relies on user input we implemented filters and exceptions (as instead of text box we used the drop down box with the data binding so that the user can select the component from it instead of typing the name manually which may lead to an error) to ensure that during the use of application the database will remain consistent and no errors will occur.
- Connection handling – with the use of the external configuration file, it is extremely easy to change the database connection and transfer our program onto different systems. Beside this one, we implemented strict control of connection and secured it with a reasonable amount of exceptions.
- User authorization and tracking – The application identifies users with the use of the logging window and runs the proper application. Additionally, the cashier window is extended with the tracking of the logged user so that it is possible to control who sold s particular ticket and later on use this relation in the statistics tool.

6 GRAPHICAL USER INTERFACES DESIGN

6.1 Login Window



Figure 7. Login Window.

This is the first window created with Expression Blend. As it is the first point at which the user actually experiences our program, we wanted to make it as customized and as characteristic as possible. To achieve this, we invented an interesting visual design and connected it with some appealing animations.

This is also the moment where we established a connection with the database. On the basis of given username and password, the type of user is obtained and the corresponding window is invoked.

6.2 Admin Window

After logging in as the administrator, we will see Admin Panel View.

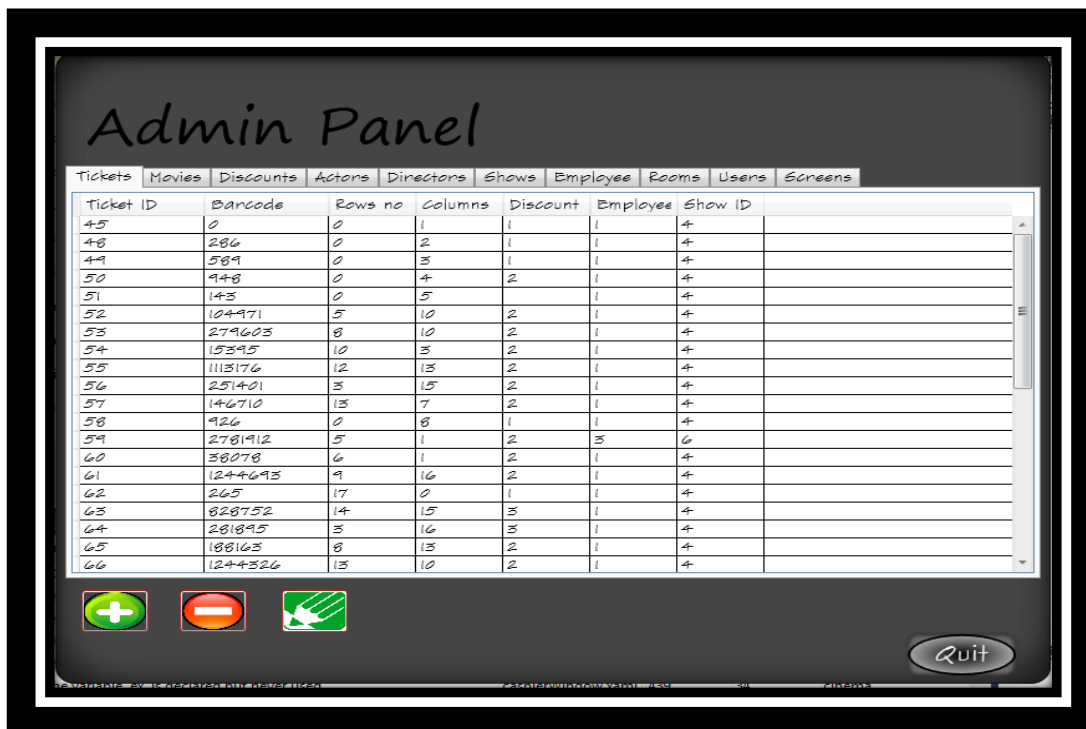


Figure 8. Starting Administrator Panel.

The administrator has got the full right on the database, and can modify, add, and delete rows in each our table. Let's focus on one table, in this case, the Users table. After changing the tab, we can see a "DataGrid" with data from the Workers table.[10]

To load a data set, we use a LINQ query, and add Binding to each column in XAML.[11] Bindings connect our asset with objects. Objects are transferred to our "DataGrid" and are assigned to a particular column. XAML interprets by field name in object and enters it to the bind column.[1] In C# code we assign our collection to "ItemsSource" attribute of our grid.

User ID	Login	Password	Type	FirstName	LastName
100	admin	admin	a	Aman	Yadav
101	cashier1	cashier1	c	Piotr	Sadlo
102	cashier2	cashier2	c	Krzysztof	Olchowik
105	manager	manager	m	lol	lolek

Figure 9. Worker's Table.

Table 2. Binding updates Grid in C#.

```

<TabItem x:Name="tabUsers" Header="Users" FontFamily="SketchFlow Print" FontSize="13.333">
  <Grid Background="#FFE5E5E5">
    <DataGrid x:Name="usersDataGrid" AutoGenerateColumns="False" Margin="0,0,8,8">
      <DataGrid.Columns>
        <DataGridTextColumn Header="User ID" Width="100" Binding="{Binding idUser}"/>
        <DataGridTextColumn Header="Login" Width="100" Binding="{Binding name}"/>
        <DataGridTextColumn Header="Password" Width="100" Binding="{Binding password}"/>
        <DataGridTextColumn Header="Type" Width="100" Binding="{Binding type}"/>
        <DataGridTextColumn Header="FirstnName" Width="100" Binding="{Binding firstname}"/>
        <DataGridTextColumn Header="LastName" Width="100" Binding="{Binding lastname}"/>
      </DataGrid.Columns>
    </DataGrid>
  </Grid>
</TabItem>

```

As mentioned before, our Administrator Panel can add, modify, and edit the rows in Database. Let's take a look at the Add function. After choosing a specific tab, and pressing the Add button, immediately appears animation (a unique animation that depends on the tab which we chose with our Add Window). This animation is generated by using Blend and XAML Animation Code.

Table 3. XAML Animation Code.

```

<Storyboard x:Key="UserGrid">
  <DoubleAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.RenderTransform).(TransformGroup.Children)[3].(TranslateTransform.X)"
    <EasingDoubleKeyFrame KeyTime="0" Value="0"/>
    <EasingDoubleKeyFrame KeyTime="0:0:0.7" Value="258.40000000000003"/>
    <EasingDoubleKeyFrame KeyTime="0:0:2" Value="666"/>
  </DoubleAnimationUsingKeyFrames>

```

To change the animate parameters, we should change the values of Key Time and Value attribute. These two attributes inform us about special location (Value) and time (Key Time) when this attribute is at a particular location. Moreover, the first Key Time="0" is a time when the animation starts, and last, in this case, the Key Time="0:0:2" the time when animation finishes.

This code shows the dynamic start of our Animation. At first, we have to find our object which is responsible for our Animation. To do this, we have to look into our resources and find what ? (by name) that is interesting us. Here we used casting, because there are variable types of resources, and we are looking for the Storyboard object, which is responsible for our animation. The calling method Begin () is starting our animation.

Table 4. Code to start Animation dynamically.

```

else if (selectedTab.Header.Equals("Users"))
{
  int numb = con.Workers.Max((u => u.idUser)) + 1;
  userIdTextBox.Text = numb.ToString();

  myStoryboard = (Storyboard)this.Resources["UserGrid"];
  myStoryboard.Begin();
}

```

Figure 11. Add window, filling rows.

Figure 10. Add window.

After filling our rows and pressing the accept button, we invoke the event, which will assign particular texts from text boxes to particular attributes of our object and use the LINQ function to add a new row to the database. We also used some constraints, which are protecting us from writing wrong strings. In this case, the function Trim () checks if textbox is not empty, and the function Try Parse () protects us from typing letters instead of numbers. After this, we invoke the update function, and we can see that our grid is updating immediately with a new row.

User ID	Login	Password	Type	FirstnName	LastName
100	admin	admin	a	Aman	Yadav
101	cashier1	cashier1	c	Piotr	Sadlo
102	cashier2	cashier2	c	Krzysztof	Olchowik
105	manager	manager	m	lol	lolek
106	admin1	admin1	a	Jan	Kowalski

Figure 12. Table after updating with new row.

In order to update the row the function insertOnSubmit() invokes a simple query INSERT INTO" in our Database which in this case is:
 INSERT INTO workers VALUES('admin1','admin1','Jan','Kowalski','a');

Table 5. C# code, adding rows, constraints.

```

private void userAcceptEvent(object sender, System.Windows.RoutedEventArgs e)
{
    if (eventFlag == true)
    {
        float number;
        user = new Worker();

        if (userFirstNameTextBox.Text.Trim() == "" || userLastNameTextBox.Text.Trim() == "" || userPasswordTextBox.Text.Tr
            MessageBox.Show("Fill all fields!");
        else if (float.TryParse(userLastNameTextBox.Text, out number) == true || float.TryParse(userFirstNameTextBox.Text,
            MessageBox.Show("Fill correct values!");
        else
        {
            user.firstname = userFirstNameTextBox.Text.Trim();
            user.lastname = userLastNameTextBox.Text.Trim();
            user.password = userPasswordTextBox.Text.Trim();
            user.name = userLoginTextBox.Text.Trim();

            ComboBoxItem item = userTypeComboBox.SelectedItem as ComboBoxItem;
            if (item.Content.ToString().Equals("Cashier"))
            {
                user.type = "c";
            }
            else if (item.Content.ToString().Equals("Admin"))
            {
                user.type = "a";
            }
            else if (item.Content.ToString().Equals("Manager"))
            {
                user.type = "m";
            }
        }
        con.Workers.InsertOnSubmit(user);
    }
}

```

Let us take a look at another function – deleting rows from the database. At first, we should select a row which we want to delete. If we will not do it, then the program will send us a warning. After clicking a button, we invoke an event, which runs the LINQ query to find object which we want to remove, and after that, we use the built-in method `DeleteOnSubmit()`. In query, we used the method `FirstOrDefault()` which is responsible for returning only one object, not a collection of objects. Finally, object is removed.

102	cashier2	cashier2	c	Krzysztof	Olchowik
105	manager	manager	m	lol	lolek
106	admin1	admin1	a	Jan	Kowalski

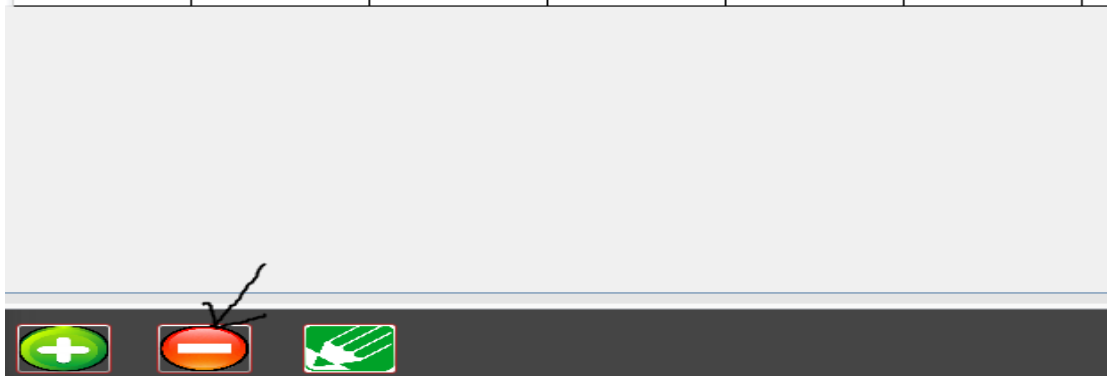


Figure 13. Deleting a row.

Table 6. C# code of deleting rows.

```

else if(selectedTab.Header.Equals("Users"))
{
    userSelected = usersDataGrid.SelectedItem as Worker;

    if(userSelected == null)
    {
        MessageBox.Show("Choose record to delete!");
    }
    else
    {
        Worker use = (from us in con.Workers where us.idUser == userSelected.idUser select us).FirstOrDefault();
        con.Workers.DeleteOnSubmit(use);
        SubmitChanges();
        UpdateUserDataGrid();
    }
}
}

```

In the Admin view, we add one more function – modify records. To modify a record, we have to select the record which is the most interesting for us and click the button below to call the event.

105	manager	manager	m	lol	lolek	
106	admini	admini	a	Jan	Kowalski	

Figure 14. Selecting Row.

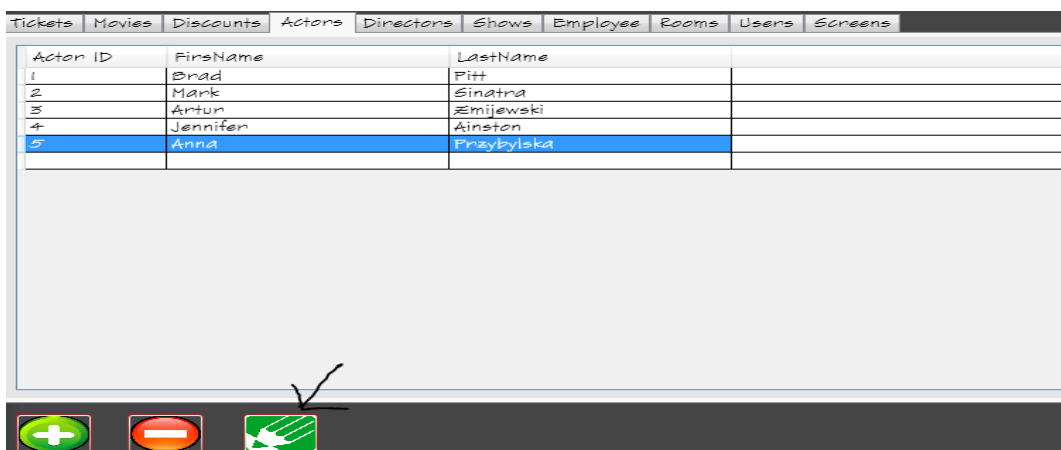


Figure 15. Modify rows.

There is what? called an animation with window. It is the same as add event, but with filled rows. In the code below, our Program is casting a row type "DataGridItem" to Actor. Then we used the attributes of the Actor object to fill all the fields. We are using the LINQ query statement to find our object in mapped objects, and then we can change the fields.

Table 7. C# code fills fields.

```

else if (selectedTab.Header.Equals("Actors"))
{
    Actor actor = actorsDataGrid.SelectedItem as Actor;

    if (actor == null)
    {
        MessageBox.Show("Choose record to modify!");
    }
    else
    {
        myStoryboard = (Storyboard)this.Resources["ActorGrid"];
        myStoryboard.Begin();
        actorIDTextBox.Text = actor.actor_id.ToString();
        actorLastName.Text = actor.surname;
        actorFirstNameTextBox.Text = actor.name;

        actorToEdit = new Actor();
        actorToEdit = (from act in con.Actors where act.actor_id == actor.actor_id select act).FirstOrDefault();
    }
}
}

```

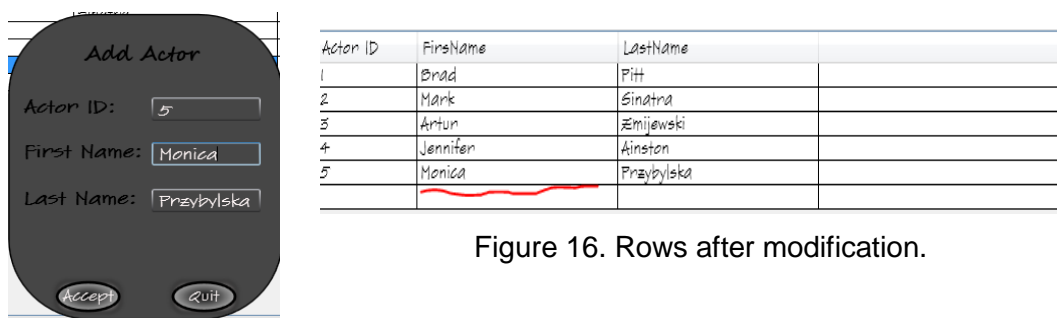


Figure 16. Rows after modification.

Figure 17. Windows with filled fields.

After changing fields and clicking the modify button, an event is called which uses LINQ and changes our record in the database. Then the values are assigned from the text box fields to the attributes of our object, which will then call a method `submitChanges()` (works like `MODIFY` in SQL). This is the built-in LINQ method which accepts the changes in object and maps it into database. Actually this is the same event as after clicking the add button but the program performs a different function. The program sets a flag event called `eventFlag` which shows which part of code should be invoked.

Table 8. Modifying record C# and saving it into database.

```

else if (eventFlag == false)
{
    actorToEdit.actor_id = Int32.Parse(actorIDTextBox.Text.Trim());
    actorToEdit.name = actorFirstNameTextBox.Text.Trim();
    actorToEdit.surname = actorLastName.Text.Trim();

    SubmitChanges();
}
UpdateActorDataGrid();
}
}

```

In the program we also used the binding text template in order to display our data in Combo Boxes which have already been bound with the database. Several attributes with various parameters and binding sets are assigned to a employee template where X:Key is the name of our template. In places where we are declaring our component, we used the Item Template parameter to load our template to items. Static Resource emphasizes that template is static, and we cannot change it dynamically.

Table 9. XAML template code.

```
<DataTemplate x:Key="employeeTemplate">
    <StackPanel Margin="5 5 5 0" Orientation="Horizontal">
        <TextBlock FontFamily="SketchFlow Print" FontSize="8" Foreground="LightGray" Text="Employee ID: "/>
        <TextBlock FontFamily="SketchFlow Print" FontWeight="Bold" FontSize="9" Foreground="IndianRed" Text="{Binding Path=emp_id}"/>
        <TextBlock FontFamily="SketchFlow Print" FontSize="8" Foreground="LightGray" Text=", Name: " />
        <TextBlock FontFamily="SketchFlow Print" FontWeight="Bold" FontSize="9" Foreground="IndianRed" Text="{Binding Path=name}"/>
        <TextBlock FontFamily="SketchFlow Print" FontSize="8" Foreground="LightGray" Text=" " />
        <TextBlock FontFamily="SketchFlow Print" FontWeight="Bold" FontSize="9" Foreground="IndianRed" Text="{Binding Path=surname}"/>
    </StackPanel>
</DataTemplate>
```

Table 10. Using template in Combo Box component.

```
<ComboBox x:Name="showEmployeeComboBox" Margin="90.833,135.167,18.499,0" Foreground="White" Background="{x:Null}" Height="16.667"
    VerticalAlignment="Top" Grid.ColumnSpan="2" SelectionChanged="showEmployeeEvent" ItemTemplate="{StaticResource employeeTemplate}"/>
```



Figure 18. Changing window.

Table 11. C# Deleting items from listbox using key.

```
private void movieShowDelete(object sender, System.Windows.Input.KeyEventArgs e)
{
    ListBox box = sender as ListBox;

    Actor boxItem = box.SelectedItem as Actor;

    if (e.Key == Key.Delete)
    {
        actorToListBox.Remove(boxItem);
        movieActorListBox.ItemsSource = null;
        movieActorListBox.ItemsSource = actorToListBox;
    }
}
```

As we can see in order to account for the many to many relationship(M:N), we used list boxes. Clicking on the combo box calls an event and the (?) item is added to the listbox. We can add some items from the combo box. We can also remove items from the combo box after selecting an item. Clicking the “delete” key, then calls the function remove(object) to delete it from the combo box. Finally to refresh the listbox, the collection of objects are assigned to the attributes of listbox.

6.3 Manager Window

The Manager Window is our tool for statistics. It can filter data by date and pick values that will be plotted later on in an Excel chart. The general idea for this application is the future development possibilities as the GUI loads an external dll(type of a file) file providing all the logic functions. Parts encoded in this application only obtain dates from date pickers and choose paths for saving a file. For this purpose, we use an in-built Windows Dialog Form that has the full support for system functions and exceptions. The rest of the functionality is based on external dll that loads values for combo-boxes and with the button 'Save', it executes the proper part of the code. In a real life situation, this kind of solution has lots of advantages. Thanks to this, our product can be installed on a client system and later on when the client requests more analysis functions, we can provide them by changing only one file. So the already installed version does not have to be changed, only the external file, loaded during runtime.

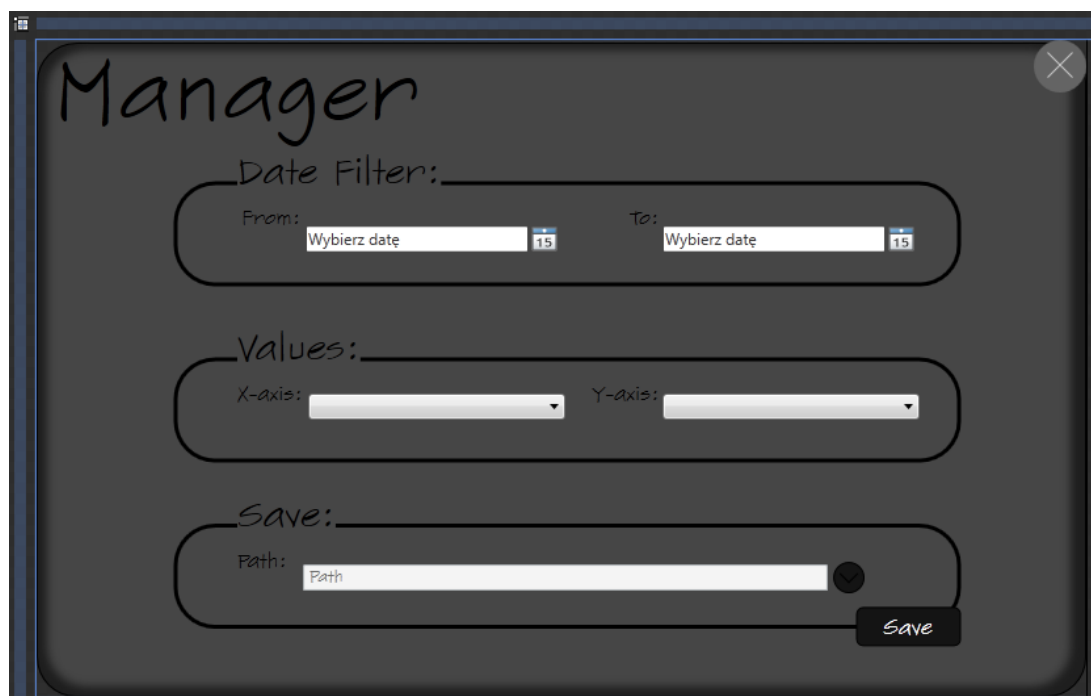


Figure 19. Manager Window.

The figure above presents the outlook of statistics tool where most of the controls are created manually or at least have an applied template. This appearance fits in our application design and main idea of style. Although it looks minimalistic, it offers a lot of options which can be still extended.

Table 12. Code for creating the path.

```
private void pathButton_Click(object sender, System.Windows.RoutedEventArgs e)
{
    System.Windows.Forms.FolderBrowserDialog pathBox = new System.Windows.Forms.FolderBrowserDialog();
    pathBox.ShowDialog();
    pathTextBox.Text = pathBox.SelectedPath;
}
```

Before presenting the code of this application, it is worth mentioning the following:

- Use of external dll file:
At first, we added an additional project to our solution in the form of Class Library, then we created a reference between the main part and the one that is responsible for obtaining dll.[3]

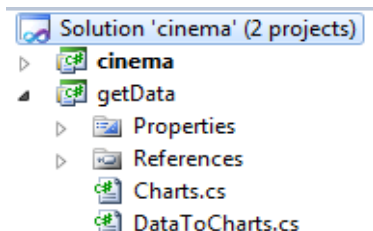


Figure 20. Additional Project.

- Then it was quite simple to refer to functions that are represented physically as dll file. Using the 'using' directive manager window, it was joined with the dll project and which in code can be used as follows:

Table 13. Code to represent dll file.

```
Charts obj = new Charts(pathTextBox.Text, yAxisData, xAxisData, xAxisName, yAxisName);
obj.DrawChart();
```

- Invoking window for choosing the path to saving excel charts:
Since we needed two methods that can build a content of combo boxes and exports data to excel, we picked this way of loading dll. Any additional changes can be introduced by change of code in the getData project, building it individually and then replacing the getData.dll file in the folder where the cinema.exe file is placed. It is a must to mention what the dll looks like, because it uses additional features from Office Package but also fills in combo-boxes in the user interface:

Table 14. Code to show how our Excel chart should look like.

```

public List<String> getDataAxisY(String axX)
{
    String axisX = axX;
    List<String> dataToComboBox = new List<String>();
    String comboItem2;
    String comboItem1;

    String a = axX;

    if (axisX.Equals("Movies"))
    {
        comboItem1 = "Tickets";
        comboItem2 = "Tickets-Type";
        dataToComboBox.Add(comboItem1);
        dataToComboBox.Add(comboItem2);
    }
    else if (axisX.Equals("Shows"))
    {
        comboItem1 = "Tickets";
        comboItem2 = "Tickets-Type";
        dataToComboBox.Add(comboItem1);
        dataToComboBox.Add(comboItem2);
    }
}

```

- As mentioned previously, features of Office package are possible to use by means of reference and used namespace:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Excel = Microsoft.Office.Interop.Excel;
namespace getData
{
    public class Charts
    {}
}

```

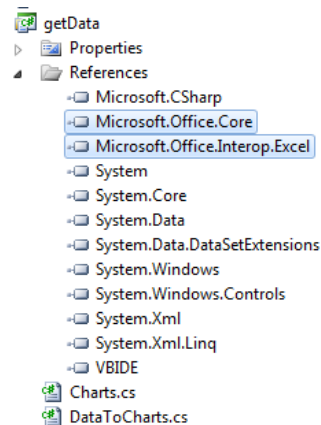


Figure 21. Using the references and namespace.

- The code to use the references and namespace is as follows:

Table 15. The code to use references and namespace.

```

public void DrawChart()
{
    Excel.Application xlApp = new Excel.Application();
    xlApp.Workbooks.Add();
    Excel.Worksheet workSheet = xlApp.ActiveSheet;
    workSheet.Cells[1, 1] = "";
    /* */
    Excel.Range chartRange;

    Excel.ChartObjects xlCharts = (Excel.ChartObjects)workSheet.ChartObjects(Type.Missing);
    Excel.ChartObject myChart = (Excel.ChartObject)xlCharts.Add(10, 80, 300, 250);
    Excel.Chart chartPage = myChart.Chart;

    Excel.Range last = workSheet.Cells.SpecialCells(Excel.XlCellType.xlCellTypeLastCell, Type.Missing);
    chartRange = workSheet.get_Range("A1", last);
    chartPage.SetSourceData(chartRange, System.Reflection.Missing.Value);
    chartPage.ChartType = Excel.XlChartType.xlColumnClustered;

    workSheet.SaveAs(string.Format@"{0}\\" + title1 + " " + title2 + ".xlsx", path));
    xlApp.Quit();
}

```

The presented solutions are in agreement with the theory of using Microsoft Office Excel features. In the beginning an object of an application was created, and then we extended it with workbooks and worksheets. Having that implemented, it is possible to treat a worksheet as an array and point to particular indexes and assign values. Later on, we build a chart and indicate a range of labels and data. To ensure that the range is always correct, we use a function that returns the position of the last used cell. In the end, the file is saved with titles passed from the user interface so that it is clear what kind of data is included in the chart.

6.4 Cashier Window

In the Cashier View, user has fewer rights than admin. He can only take a look at some pictures. After logging to Cashier View, we will enter a window with “Data Grid”, empty ticket fields and some buttons which the program uses for filtering information, and to display data in the “DataGrid” components, we used the Linq query to get Data Set then binding in XAML (in the same way as in Admin Panel) to display data, and after that, we set Data Set to the Data Grid using attribute ItemSource of “DataGrid” Component. As we can see here, we used data from few tables, and used the join condition. To properly bind and display data, we created another class, MixedData, by using mixed attributes to object, and then assigning MixedData collection to ItemsSource in order to bind it.

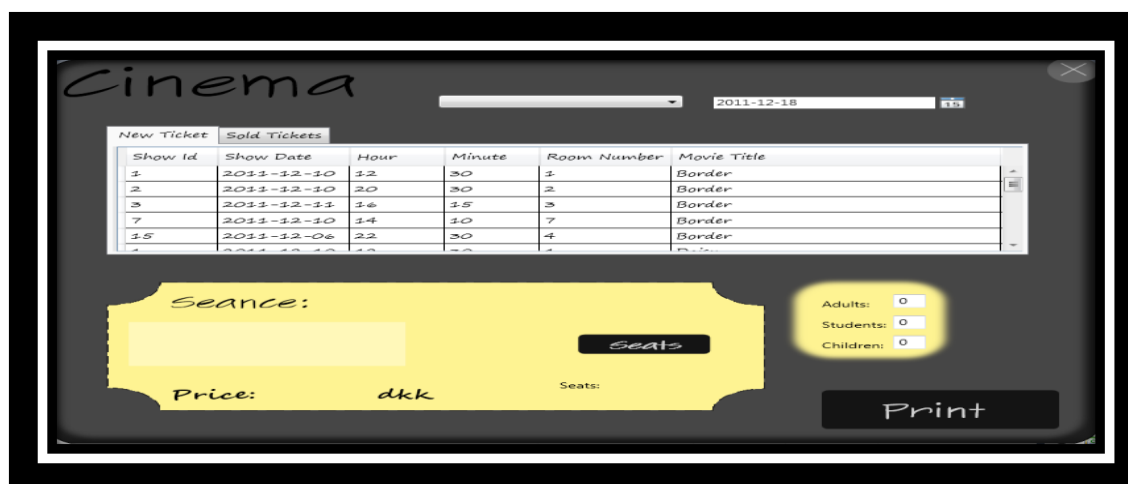


Figure 22. Start window cashier.

Table 16. Updating data grid, binding, linq queries.

```

private void UpdateDataGrid()
{
    var shows = (from show in con.Shows
                 join showMovie in con.Show_Movies
                 on show.show_id equals showMovie.show_id
                 join movie
                 in con.Movies on showMovie.movie_id equals movie.movie_id
                 select new { show.show_id, show.date, show.room_id, show.hour, show.minute, movie.title }).ToList();

    //SQL Statement
    /*
    SELECT s1.show_id, s1.Date, s1.room_id, m1.Title
    FROM Show s1, Movie m1, Show_Movie s2
    WHERE (s1.show_id = s2.show_id and s2.movie_id= m1.movie_id)
    ORDER by show_id;
    */

    List<MixedData> mixedData = new List<MixedData>();

    foreach (var item in shows)
    {
        MixedData helpedMixedData =
            new MixedData(item.show_id, item.hour, item.minute, item.date, Int32.Parse(item.room_id.ToString()), item.title);
        mixedData.Add(helpedMixedData);
    }

    TicketDataGrid.ItemsSource = mixedData;
}

```

The Date Picker and combo Box component are responsible for filtering our data in the grid. In DatePicker, we added one or more functionality, that we cannot choose date, before the present day. Choosing past data is useless in our case. After running a program, the Date Picker always displays a current date, so we write from the Date Picker textField date, split it using the function Split()(to get day, month, year), and put our strings to the method SetBlackOutDates() where we set CalendarDateRange into BlackoutDates attribute. We should notice that we decrement day by 1, because we want to select the current day, but we do not want to select days before. To filter data, we use the LINQ query with the chosen attributes and condition WHERE (like in SQL) then assign a collection of data to DataGrid attribute Items Source.

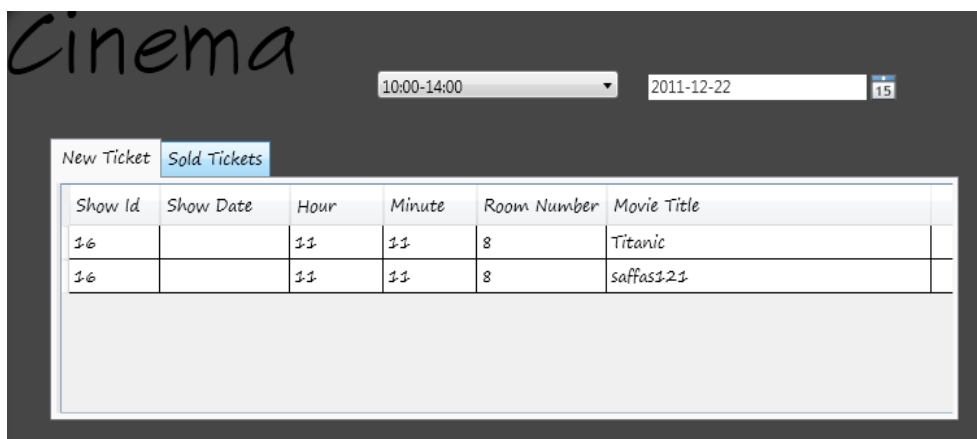


Figure 23. Filtered data and displayed in the grid.



Figure 24. DatePicker with BlackOutDays

Table 17. C# code which writes and splits current data (month, day, year).

```

ComboBoxItem filtr = (ComboBoxItem)timePicker.SelectedItem;

if (!datePicker.Text.Equals(""))
{
    String[] dateSplitted = datePicker.Text.Split('-');

    DateTime date = new DateTime(Int32.Parse(dateSplitted[0]), Int32.Parse(dateSplitted[1]), Int32.Parse(dateSplitted[2]));
  
```

Table 18. C# code which sets blackout dates.

```

public void SetBlackOutDates(int year, int month, int day)
{
    datePicker.BlackoutDates.Add(new CalendarDateRange(
        new DateTime(2000, month, day - 1),
        new DateTime(year, month, day - 1)));
}
  
```

CalendarDateRange.CalendarDateRange
Initializes a new instance of the System.

Table 19. Filtering data in C#.

```

else if (time.Equals("14:00-18:00"))
{
    TicketDataGrid.ItemsSource = null;

    var filteredData = (from show in con.Shows
                        join showMovie in con.Show_Movies
                          on show.show_id equals showMovie.show_id
                        join movie
                          in con.Movies on showMovie.movie_id equals movie.movie_id
                        where show.hour >= 14 && show.hour <= 18 && show.date == date
                        select new { show.show_id, show.date, show.room_id, show.hour, show.minute, movie.title }).ToList();

    TicketDataGrid.ItemsSource = filteredData;
}

```

After selecting a row in the dataGrid, an event is called which will fill dynamically our Tickets. At first, the program cast selected an item from MixedData, then read each attribute of the casted object and filled it to the ticket. We added a condition to add number '0' to minutes, because we cannot set to the database the int value which is starting from '0', so we are doing it dynamically in c# code.

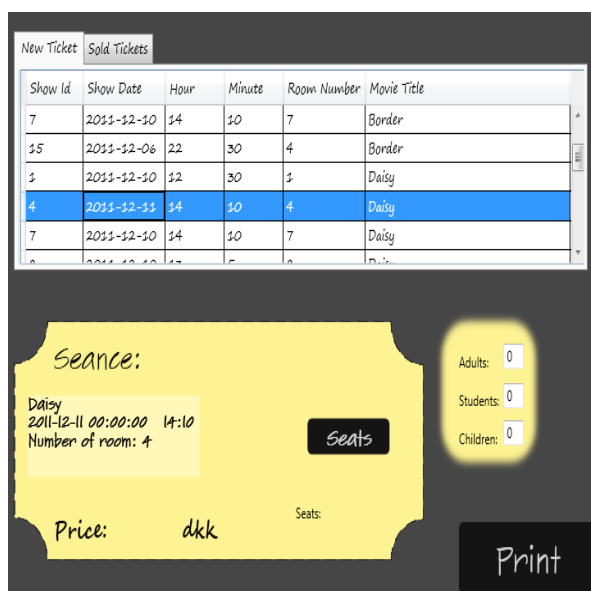


Figure 25. Dynamically filling ticket.

Table 20. C# code of a dynamically filled ticket.

```

private void selectItem(object sender, System.Windows.Controls.SelectedCellsChangedEventArgs e)
{
    if (eventFlag == false)
    {
        MixedData selectedData = TicketDataGrid.SelectedItem as MixedData;

        movieTitle = selectedData.title;
        movieDate = selectedData.date.ToString();
        movieTimeHour = selectedData.hour.ToString();
        movieTimeMinute = selectedData.minute.ToString();
        numberOfRoom = selectedData.room_id.ToString();
        movieShow_id = selectedData.show_id.ToString();

        if (movieTimeMinute.Length == 1)
        {
            ticketDetails.Text = movieTitle + "\n" + movieDate + " " + movieTimeHour + ":"
                + movieTimeMinute + "0" + "\n" + "Number of room: " + numberOfRoom;
        }
        else
        {
            ticketDetails.Text = movieTitle + "\n" + movieDate + " " + movieTimeHour + ":"
                + movieTimeMinute + "\n" + "Number of room: " + numberOfRoom;
        }
    }
}

```

Now this is the best time to choose a seat. This part of code is the most complicated because we have to set some logic to draw grid with seats, select busy places and save it to databases. After clicking on the seats button, there will appear a window similar to the one below. Red dots indicates the seats that are already reserved, green for free, and blue for chosen for us.

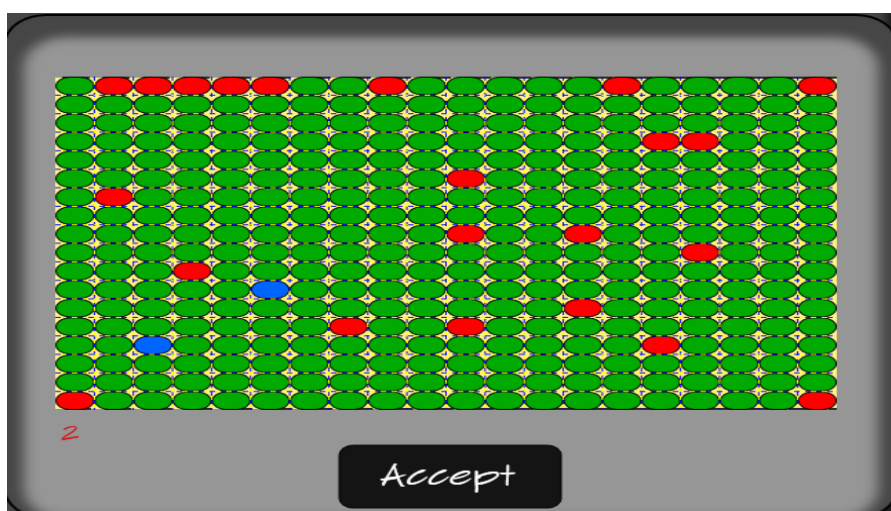


Figure 26. Window with free and busy seats.

Table 21. Linq, choosing occupied seats.

```
private void seatsEvent(object sender, System.Windows.RoutedEventArgs e)
{
    var occupiedSeats = (from show in con.Shows
                        join ticket in con.Tickets
                        on show.show_id equals ticket.show_id where show.show_id == Int32.Parse(movieShow_id)
                        select new { ticket.row, ticket.col }).ToList();

    List<TakenIndexes> takenIndexes = new List<TakenIndexes>();

    foreach (var item in occupiedSeats)
    {
        TakenIndexes seat = new TakenIndexes(item.row, item.col);
        takenIndexes.Add(seat);
    }
}
```

Table 22. Creating grid with seats.

```
for (int r = 0; r < rows[0].rows; r++)
{
    seatsGrid.RowDefinitions.Add(new RowDefinition());
}

for (int c = 0; c < rows[0].columns; c++)
{
    seatsGrid.ColumnDefinitions.Add(new ColumnDefinition());
}

seatButton button = null;
for (int i = 0; i < seatsGrid.RowDefinitions.Count; i++)
{
    for (int k = 0; k < seatsGrid.ColumnDefinitions.Count; k++)
    {
        if (takenIndexes.Count != 0)
        {
            for(int j = 0; j<takenIndexes.Count; j++)
                if ((takenIndexes[j].getRow() != i) || (takenIndexes[j].getColumn() != k))
                    button = new seatButton(i, k, false, this);
            else
            {
                button = new seatButton(i, k, true, this);
                break;
            }
        }
        else
            button = new seatButton(i, k, false, this);
        button.SetValue(Grid.RowProperty, i);
        button.SetValue(Grid.ColumnProperty, k);
        seatsGrid.Children.Add(button);
    }
}
```

When we are creating a grid with seats, at first, we have to know which seats are occupied, and how many seats are free, when the show is on. To know that, we use a LINQ query; there's a class TakenIndexes, which creates objects of taken seats (row and column). If we know which seats are occupied and now we have to know how many seats (how many columns and rows) are in the room where the show will take place. To find out this, we make another LINQ query from the Room Table. The number of seats depends on the room where show will be performed. Now we can create seats by inheriting the seatsButton from System class. Windows.Control.Button, so our seats are buttons. While the program is putting objects into our grid, it always checks the occupied indexes collection to check if our seat is free or not. To do that, the program is using a loop, and checks one by one each object in collection and tries to compare indexes, grids, and occupied seats. If the number of row and column is the

same as the one in the object of the collection, then program creates objects with appropriate parameters(flag true when busy, false when free). Each button uses a template (green, red, blue) which is defined in XAML, and then when the seat is busy, we use template "takenSeat" and when is free we use "normalSeat". To see the Resource from CashierWindow class, we transfer everything when we call a constructor. Template change depends on the position of clicked seats. After clicking, we read what the name of chosen button template is set and then we change from one template to another (from clickedSeat to normalSeat and other way). And then on clicking the buttons, we change flag of selected buttons(by changing the color) to know which buttons are clicked.

Table 23. Clicking the button dynamically changes the seats template.

```
void seatButtonClick(object sender, System.Windows.RoutedEventArgs e)
{
    if (this.Template == (System.Windows.Controls.ControlTemplate)parent.Resources["normalSeat"])
    {
        this.Template = (System.Windows.Controls.ControlTemplate)parent.Resources["clickedSeat"];
        this.clicked = !this.clicked;
        this.selected = true;
    }
    else if (this.Template == (System.Windows.Controls.ControlTemplate)parent.Resources["clickedSeat"])
    {
        this.Template = (System.Windows.Controls.ControlTemplate)parent.Resources["normalSeat"];
        this.clicked = !this.clicked;
        this.selected = false;
    }
}
```

After choosing seats, and clicking on the accept button, the program checks out if number of clicks is appropriate (depending on number of tickets which we choose in textbox fields in tickets), and will look up into each button (seatGrid.Children[i] in for a loop in the grid to find the selected buttons. If it finds them, then it will add them to the list of clicked buttons. This list will be used later to display the chosen seats in the ticket and make appropriate queries to create a ticket. Moreover, the program will calculate the price for the ticket using LINQ queries to calculate the price for the particular ticket.

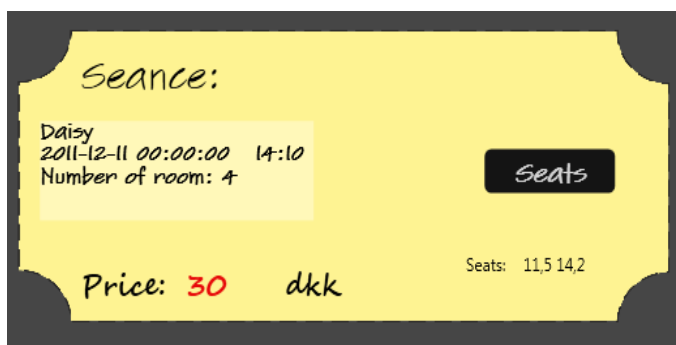


Figure 27. Ticket with chosen seats and price.

Table 24. Creating a list of selected seats.

```
double roomPrize = (from roo in con.Rooms
                    where roo.room_id == Int32.Parse(numberOfRoom)
                    select roo.price).FirstOrDefault();

prizeTextBlock.Text = ((students * discountValueStudent + children * discountValueChild + adults * discountValueAdult) * roomPrize).ToString();
numberOfClicks = 0;
seatsGrid.Children.Clear();
seatsGrid.ColumnDefinitions.Clear();
seatsGrid.RowDefinitions.Clear();
```

To make a query which will add our ticket to the database, we need to fill some additional fields. These are barcode and discountID. For each chosen seat (for each loop), the program uses Random class and method Next () to generate a random number. To get a second one, we use the LINQ query one more time. As we can see in the SoldTickets tab, our DataGrid is updated dynamically after adding the data to the database: the SoldTickets tab has two more rows with tickets which we created.

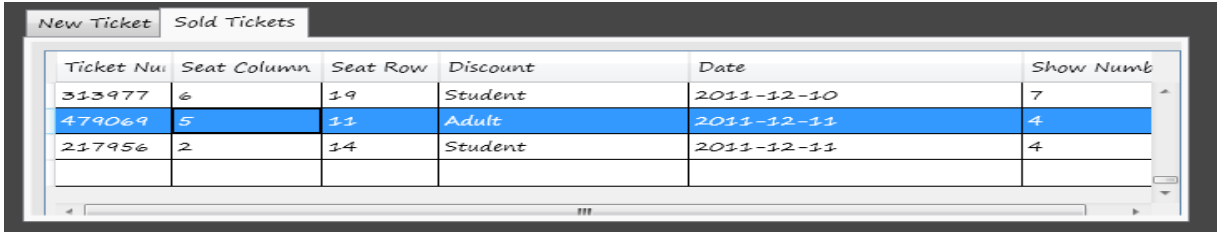
Table 25. Find rest of attributes to make a LINQ add Ticket query.

```
for (int i = 0; i < this.seatsList.Count; i++)
{
    int row = seatsList[i].getRow();
    int column = seatsList[i].getCol();

    Random random = new Random();
    int barcode = random.Next(row * column * 10000 + 1000);

    ticket = new Ticket();
    ticket.barcode = barcode;
    ticket.row = row;
    ticket.col = column;
    ticket.show_id = Int32.Parse(movieShow_id);
    ticket.emp_id = 1;

    int discountCount = (from disc in con.Discounts select disc).Count();
    for (int k = 0; k < discountCount; k++)
    {
        if (children != 0)
        {
            discountType = "Child";
            ticket.discount_id = (from disc in con.Discounts where disc.type == discountType select disc.discount_id).FirstOrDefault();
            children--;
            break;
        }
        else if (adults != 0)
        {
            discountType = "Adult";
            ticket.discount_id = (from disc in con.Discounts where disc.type == discountType select disc.discount_id).FirstOrDefault();
            adults--;
            break;
        }
        else if (students != 0)
        {
            discountType = "Student";
            ticket.discount_id = (from disc in con.Discounts where disc.type == discountType select disc.discount_id).FirstOrDefault();
            students--;
            break;
        }
    }
}
```



Ticket Num	Seat Column	Seat Row	Discount	Date	Show Num
313977	6	19	Student	2011-12-10	7
479069	5	11	Adult	2011-12-11	4
217956	2	14	Student	2011-12-11	4

Figure 28. Sold Tickets Data Grid with new added tickets.

7 DATABASE MODULES

7.1 Tables

The following sections will describe the process of creating the database with the necessary tables, relations and other features.

7.1.1 Property Tables

A property table was made in order to have an overview of the different tables, .thus making it a lot easier to keep track of the tables changes.

Table 26. Property Table.

Table name	Column name	Domain (intended)	Example	Constraint	FK referenced table
Ticket	idTicket	int	1	PK, NOT NULL	Ref.
	row	int	40	NOT NULL	Discount:
	column	int	30	NOT NULL	discount_id
	Barcode	int	22	UNIQUE, NOT NULL	Employee: emp_id Show: show_id
Employee	emp_id	int	33	PK, NOT NULL	
	name	varchar(20)	James		
	surname	varchar(200)	Bond		
Discount	discount_id	int	1	PK, NOT NULL	
	type	varchar(20)	Student		
	Value	float	50.00	NOT NULL	
Show	show_id	int	3	PK, NOT NULL	Ref.
	date	date	2012/01/02	NOT NULL	Room: room_id
	hour	int	1	NOT NULL	
	minute	int	30	NOT NULL	

Room	room_id	int	22	PK	Ref. Screen: screen_id
	Rows	int	55	NOT NULL	
	Columns	int	3	NOT NULL	
	Price	int	55	NOT NULL	
Screen	screen_id	int	3	PK	
	Type	varchar(20)	HD	UNIQUE,NOT NULL	
	Price	int	80	NOT NULL	
Movie	movie_id	int	444	PK	Ref. Director: director_id
	Title	varchar(45)	Star Wars	NOT NULL	
	Year	int	1973	NOT NULL	
	Duration	int	02:03.33	NOT NULL	
	Description	text	scifi....		
Actor	actor_id	Int	555	PK	
	Name	varchar(20)	Brad	NOT NULL	
	Surname	varchar(20)	Pit	NOT NULL	
Director	director_id	int	33	PK	
	Name	varchar(20)	Micheal	NOT NULL	
	Surname	varchar(20)	Bay	NOT NULL	
Show_movie	movie_id	int	33	PK	Ref. Show: show_id Movie: movie_id
	show_id	int	2	PK	
Employee_Show	emp_id	int	333	PK	Ref. Employee: emp_id Show: show_id
	show_id	int	2	PK	
Actor_Movie	actor_id	int	555	PK	Ref. Actor: actor_id Movie: movie_id
	movie_id	int	33	PK	

7.1.2 Table Definition

All the tables in our database are created via the following table definitions. The table definitions do not differ so much from each other so only a few well chosen are shown and describe here. The features mentioned in these table definition match with most of the other tables’.

Ticket Table Definition:

```
CREATE TABLE Ticket
(idTicket int IDENTITY(1,1) NOT NULL,
barcode int NOT NULL,
row int NOT NULL,
col int NOT NULL,
PRIMARY KEY(idTicket),
UNIQUE(barcode)
);
```

The above table contains the basic information regarding tickets, such as barcode, row, col and an id for each ticket. An explicitly a unique id has also been created to identify the tables. The foreign key relationships to other tables are added as follows:

```
ALTER TABLE Ticket
ADD discount_id int references Discount(discount_id);
```

```
ALTER TABLE Ticket
ADD emp_id int references Employee(emp_id);
```

```
ALTER TABLE Ticket
ADD show_id int references Show(show_id);
```

```
ALTER TABLE Ticket
ADD FOREIGN KEY (emp_id) references Employee(emp_id);
```

```
ALTER TABLE Ticket
ADD FOREIGN KEY (show_id) references Show(show_id);
```

```
ALTER TABLE Ticket
ADD FOREIGN KEY (discount_id) references Discount(discount_id);
```

The following table containing the relations between movies and actors is the result of a many-to-many relation between the Actor and Movie tables.

```
CREATE TABLE Actor_Movie
(actor_id int ,
movie_id int,
PRIMARY KEY(actor_id, movie_id)
);
```

This table definition was chosen primarily to describe the additional information that some of the other table definitions contains. Thus, the two examples have covered all the table definitions as all of them are similar to either one of the two table definitions.

```
ALTER TABLE Actor_Movie
ADD FOREIGN KEY (Actor_id) references Actor(Actor_id) ON DELETE
CASCADE;
```

```
ALTER TABLE Actor_Movie
ADD FOREIGN KEY (Movie_id) references Movie(Movie_id) ON
DELETE CASCADE;
```

Foreign keys are added with additional parameter, ON DELETE CASCADE. This option is used to indicate that when rows are deleted in a child table, then the corresponding rows should be deleted in the parent table. If this is not specified, the default behaviour will be that the server prevents us from deleting data in a table if other tables reference it.

7.2 Queries

At this point, we created some queries to show that our database works properly. We used a variety of SQL calluses less and more complicated.[6] The types of SQL queries are described and shown in the code below:

7.2.1 Multi Join, Where

This query is displaying tickets with prices was less than 15 dkk.

```
SELECT Ticket.barcode, room.price, room.room_id,
show.show_id, Discount.type, Discount.value,
Movie.movie_id From Discount
JOIN Ticket ON Discount.discount_id =
Ticket.discount_id
JOIN Show ON Show.show_id = ticket.show_id
JOIN Show_Movie ON Show_Movie.show_id = Show.show_id
JOIN Movie ON Show_Movie.movie_id = Movie.movie_id
JOIN Room ON Room.room_id = Show.room_id
WHERE Room.price*Discount.value >15;
```

7.2.2 Groupby, Having, Virtual Column

This query is displaying the number of tickets sold for shows, includes their type, where the number of sold tickets is larger than 4.

```
SELECT show.show_id, Discount.type,
COUNT(Ticket.idTicket) AS NumberOfTicket FROM Discount
JOIN Ticket ON Discount.discount_id =
Ticket.discount_id
```

```
JOIN Show ON Show.show_id = Ticket.show_id
GROUP BY Discount.type, show.show_id
HAVING COUNT(Ticket.idTicket) > 4;
```

7.2.3 Except, Intersect And Union Expression

Except expression - actors by name, surname.

```
SELECT * FROM Actor
EXCEPT
SELECT * FROM Director;
```

Intersect expression - actors by name, surname.

```
SELECT * FROM Actor
EXCEPT
SELECT * FROM Director;
```

Unions directors and actors - by name, surname

```
SELECT name, surname FROM Director
UNION
SELECT name, surname FROM Actor;
```

8 TESTS AND FIXES

An application can only be considered reliable and trustworthy as long as it has been subjected to pass through various tests. In our case, we paid additional attention to the correctness of our application and data by controlling and managing all data which has lead us to making sure that database is always consistent.[5] Yet, it our application still had to pass through tests as explained below.

8.1 Compatibility Testing

During this process of testing, we tried to run our application on the entire Windows (XP, Vista, Windows 7, and 8) running platform with .NET framework installed on the system and it worked perfectly fine. Whatwe could not do was to test it on the systems that came before Windows XP, simply because we could not find those systems running those OS at this present time.

8.2 Functional Testing

At this point of testing, we tried to test the functional behaviour of our application. For this, we showed the demo to the user before they could try it themselves. Here we had some run-time problems that the application faced.

8.2.1 Run-Time Problems

- The first problem was to create a connection to the database. Since our database was running on the same machine and every time we had to move to some other hosts, the database connection name (which is dependent on the host PC name and database name used) should be changed accordingly. Alternatively, the whole database is supposed to be placed on the server and the application is supposed to be run on from a client machine.
- Secondly, our application at the moment is meant to use a specific format of date and time (i.e. mm-dd-yyyy), so if the format of a user system is different from this, it will have a run-time error. In order to fix it, our system time is supposed to be changed in a specific manner, or the application is supposed to be coded with all the possible formats of date and time that the system could use.

8.3 Security Testing

For the security testing we tried to sign in with different user credentials, but login was unsuccessful. The problems are:

- At this point, our application is supposed to be slightly weak as the user can try to log in with wrong credentials as many times as they want without being traced, or let the system crash. This problem can be avoided by having a condition set in our application that could monitor the number of allowed wrong tries for a specific user and then banned the user as soon as he exceed the limits.
- Another security threat is the password that is stored in our database. It is stored in the clean text format which is a very weak point. So in order to prevent this, some encryption is supposed to be applied for it, which, unfortunately, we

could not apply in our application. Thus it has been left for the future improvements.

9 FUTURE IMPROVEMENT

After the necessary tests, we were able to fix some of the problems, but most of those problems have been left for future improvements as explained below:

- In order to avoid the database connection problem since everyone uses the same database at the same time, it is compulsory to keep the database on a server with specific credentials that could be shared by the entire clients using the application.
- In order to prevent the database from being misused, the password stored in the clean text format in our database should be encrypted.
- To prevent the run-time error caused by the date format, our application should be programmed in such a way that it could handle all the formats of date and time.
- Since there is nothing like backing up or restoring in the application, application should be implanted in order for the user to back it up and restore the system from the application itself whenever the application get crashed or affected by a virus.
- Since the application does not care for how many times someone could try to get logged in, there could be cases of lost passwords by intent, and some scammers could try to hack the system. In order to prevent unauthorized access to the system, the number of attempts with wrong credentials should be fixed. Moreover, application should monitor all the users logged in to the system with the time stamped on it.

10 CONCLUSIONS

In conclusion, the principal goal of this thesis was met; we created a desktop application for a movie theatre that could work well with a database. In the process, C# application was developed. LINQ, WPF, Office Excel and external DDL were all used as additional extensions. All the applications involved were designed so that they could have reliable interactions with users in a very convenient and efficient manner. The effectiveness and reliability of these applications were tested, and it was confirmed that the applications were capable of serving their expected purposes. The implementation of our design decisions was successful.

REFERENCES

1. DataBinding in WPF,
Available from <http://www.wpftutorial.net/DataBindingOverview.html>
(Accessed 05 January 2013)
2. Edgar, F. Codd. 1997, 'Relational Database', Available from
<http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/reldb/>
(Accessed 10 January 2013)
3. Managing solutions, Projects, and Files,
Available from [http://msdn.microsoft.com/en-us/library/cc294498\(v=expression.30\).aspx](http://msdn.microsoft.com/en-us/library/cc294498(v=expression.30).aspx)
(Accessed 05 January 2013)
4. MSDN, 'ASP .NET Overview',
Available from <http://msdn.microsoft.com/library/4w3ex9c2.aspx>
(Accessed 03 January 2013)
5. Slideshare, 'Windows/Desktop Application Testing',
Available from <http://www.slideshare.net/trupti242/window-desktop-application-testing>
(Accessed 10 February 2013)
6. SQL JOIN,
Available from http://www.w3schools.com/Sql/sql_join.asp
(Accessed 13 January 2013)
7. Why SQL Server, 'Top 12 Reasons to Choose SQL Server',
Available from <http://www.microsoft.com/en-us/sqlserver/product-info/top-twelve.aspx>
(Accessed 02 February 2013)
8. Wikipedia, 'C Sharp(Prgramming Language)',
Available from
[http://en.wikipedia.org/wiki/C_Sharp_\(programming_language\)](http://en.wikipedia.org/wiki/C_Sharp_(programming_language))
(Accessed 10 February 2013)
9. Wikipedia, 'Mircrosoft Expression Blend',
Available from http://en.wikipedia.org/wiki/Expression_Blend
(Accessed 10 February 2013)
10. WPF Tutorial, 'WPF DataGrid Control',
Available from <http://www.wpftutorial.net/DataGrid.html>
(Accessed 02 March 2013)
11. WPF Tutorial, 'XAML Editors',
Available from <http://www.wpftutorial.net/XAMLEditors.html>
(Accessed 03 March 2013)