# Database - MovieBasen

Københavns Erhvervsakademi – Software Team 9 Fall 2016

Benyam, Elvis, Mert og Casper 12/21/16 Software Development

## Contents

## Introduction *- (Mert)*

In this exam project we have decided to develop a movie web-based application similar to that of IMDB.com, where we got our inspiration from. Our application name is MovieBasen where it will be possible for users to look movies up, add movies to your profile list and review/rate them.

The application is developed in ASP.Net MVC Framework with C#, where we are going to use object-relational mapper (Entity Framework – code first) to make the database.
Our main focus in this project is the database part, where we will create a database for our application, to handle different requests. We will make some tables, and store data related to the movie concept.
We will make the database as efficient as possible, where we will try to store and retrieve data in the best way. [1]

*To see MoveBasen you can visit our site: http://moviebasen.azurewebsites.net/*

[1] *Same introduction is also used in software testing report*

## Data analysis and requirements specification *- (Mert)*

Our main users will be split in two, the normal users and the admin users, who have more of an administrator role.

**Normal User:**
A user should be able to log into the system with their username and password. They will have the option to make one if they do not have any.
Inside the system, a user should be able to search for a specific movie by the name, or by genre. The user should also be able to rate the movie (with stars or points), review it and should be able to add the movie to his/hers '*watched list*/*my movie list*'.

**Admin User:**
An admin also has the same options as the normal user, furthermore the admin should be able to make CRUD (*Create, Read, Update, Delete*) functionality to movies, genres and actors. The admin should also be able to add actor and genre to the movie. The administrator will also be able to add other users to admin-roles. The administrator should also be able to delete users that is violating the system laws/agreements.[2]

Based on these requirements we can figure out which entities there are and some of the attributes that would make sense to use:

- User - Name (char), Email Username(char), password (char)
- Movies - Movie Name (char), Year (int)
- Genres - Genre Name (char)
- Actors - Actor Name (char)
- Roles
- Rating
- Reviews
- Watched List

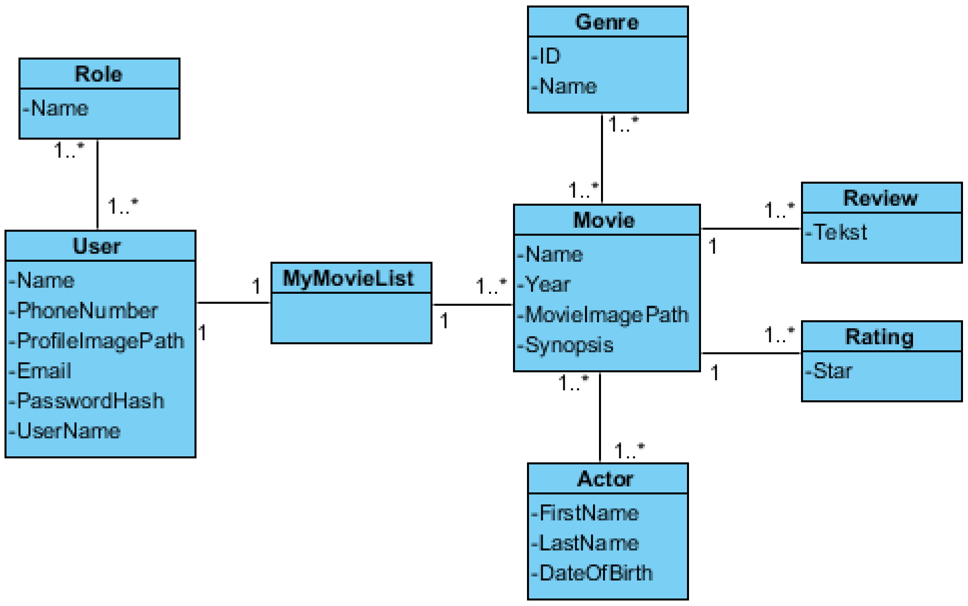
[2] Same requirements is used in software testing

## Conceptual Design *- (Elvis)*

Once you have gathered and analyzed all the business requirements, you can begin to create a conceptual design diagram for the database. In a conceptual design diagram, you describe all the entities and the different kind of relations the entities has to each other *(relationship types)*; in addition, you also describe what attributes each entity has.

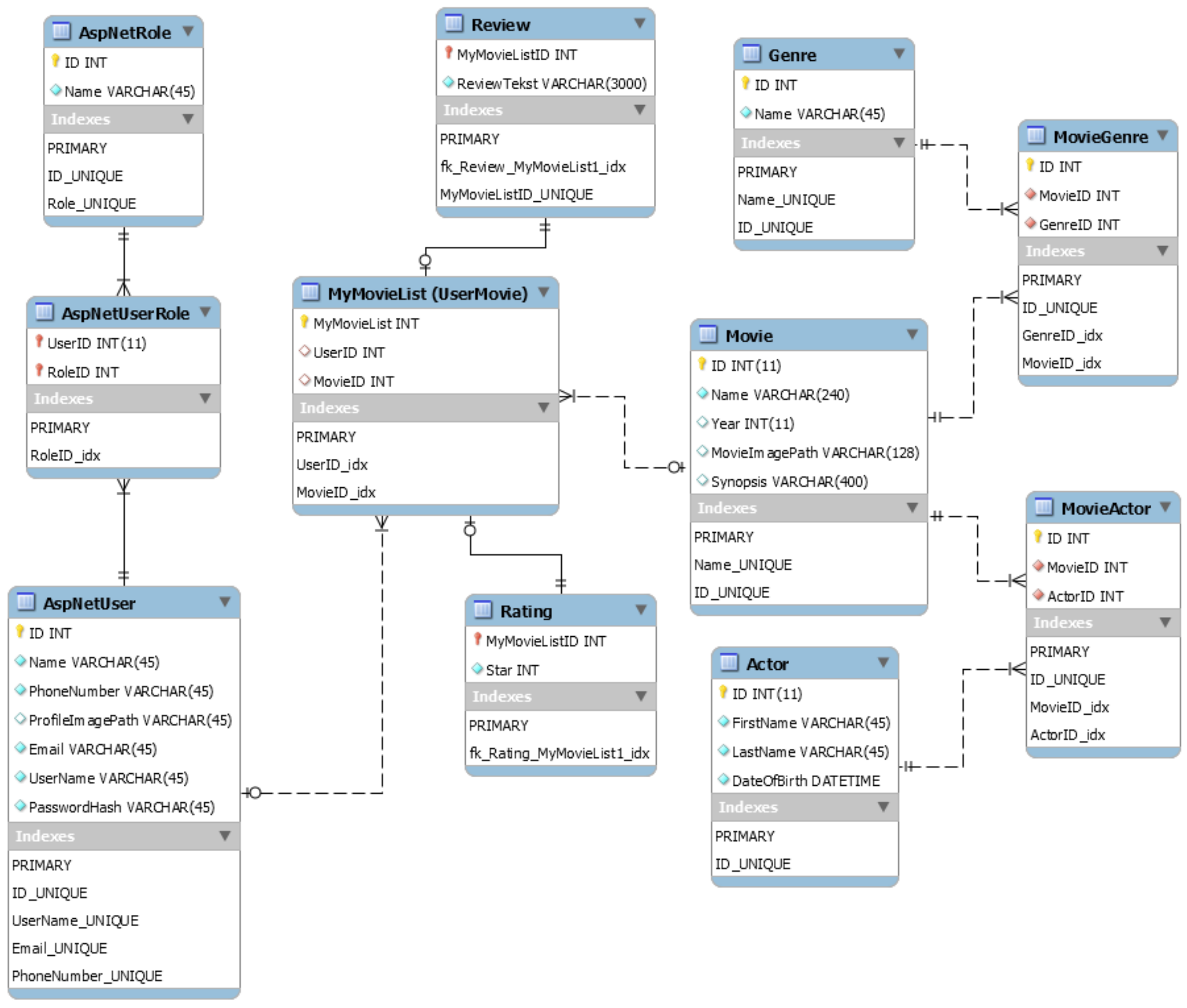
The first step of making a conceptual design is to identify all the entity and attributes. We do that by identifying all the nouns or noun in the requirements and put them in a category list. From the category list, we decide the entities and the attributes, thereafter we add associations between the entities, and give them multiplicities.

Some of the reasons to make a conceptual design is that it gives you the possibility to communicate with non-technical stakeholders, so that you can review the data model and ensure that they consider the model to be true representation of the data requirements. In addition, later on you can use it as a source of inspiration to design the Logical Design. Our conceptual design model looks like this:



## Logical Design - *(Elvis)*

After you have done the conceptual data model and the client has approved it, you can begin to make the logical data model. A logical data model describes the data in as much detail as possible *(Integrity constraint)*, and afterward you check that the database is structurally correct and able to support the required transactions. In this phase, you also normalize the database to ensure that you have eliminated redundancy and potential update anomalies. Our logical data model looks like this:



In our logical data model, we have normalized our database from normalizations rules 1 to 3, and to show our Integrity constraint you can look at our diagram. In our diagram, you can also see the primary-key, which are the yellow keys, and the foreign-key are the red keys/diamonds.
We have also in our database added cascade on delete and update on all of the table's foreign-keys, so if you delete a parent record then you will also automatically delete all of the records, in the child table.

## Physical Design *- (Elvis)*

Physical design represents the actual design blueprint of the database when you have released the system.
However, we have not made a diagram in this phase, because our database have not changed from when we made the logical design diagram *(since it would look exactly like the Logical Design diagram if we did)*.

However, what we have done is calculate what we think the approximate size of the required storage volume is going to be *(so how big is the database).* Which is;

**Movie**

| | |
|---|---|
| ID - INT | 4 Bytes |
| Name - Varchar(240) | 480 Bytes |
| Year - INT | 4 Bytes |
| MovieImagePath - VarChar(128) | 256 Bytes |
| Synopsis - Varchar(400) | 800 Bytes |
| **Total** | **1544 Bytes** |

In the movie table we expect that there's going to be 3.000.000 movies/rows in our database, that means to calculate how many byte the table size is, we calculate; tables total bytes * expected movies = 4.632.000.000 Bytes. Which means the size is 4,31 Gigabyte.
Since we cannot show all of the calculated entity/table in our database *(since we do not have room in our report)* and therefore we are only showing how we have calculated the Movie table, the rest of the calculation (*calculation of the other tables)* you can fine in the "appendix; Estimated Storage Size".

We have calculated and expect the size of our database to be 25,23 GB, and since we have chosen to launch our application on Azure server, we have found out that it's going to cost 2.856 kr pr. month for 29 GB storage space.[3]

[3] https://azure.microsoft.com/da-dk/pricing/details/sql-database/

## Entity Framework *- (Elvis)*

To make the database in our system, we have used something called Entity Framework, which is a tool you get when using Visual Studio. Entity Framework is what you call an **object-relational mapper** that enables .NET developers to work with relational data using domain-specific objects[4]. It is used when you want to connect to SQL Server and gain access to the database. The way we have used it, is by using a feature called *code-first*; the idea is that I can define classes in my model (C# objects) and then generate a database from those classes.
The reason why we have chosen to use Entity Framework's code-first approach is because; by using it you can build a database up much faster, in addition it is very easy to maintain and modify the database, and it can all be done with just a few lines of code.

**Creating a table:**

The way to create a table via the entity framework is that you first create a model class; this class contains some attributes that will represent data fields in the database table.

```
public class Movie
{
    public int ID { get; set; }
    public String Name { get; set; }
    public int Year { get; set; }
    public String Synopsis { get; set; }
    public String MovieImagePath { get; set; }
}
```

Then you go to the ApplicationDbContext class, which is located in IdentityModels.cs and you paste this line of code:

```
”public DbSet<Movie> Movies { get; set; }”.
```

What this does is that it assigns the class model to the database. The last step is to type "update database" in the console, by doing that the Framework will generate a table with the name Movie and the attributes of the class will represent database fields.

Another way to use entity framework is by using a method called "Database-First", as the name suggests, it is about you first create the database, and thereafter auto generates the code from the database.

[4] https://www.asp.net/entity-framework

## Relations between tables - (Elvis)

To create relation between tables via entity framework you need to use something called navigation properties, which provide a way to navigate an association between two entity types via classes. In our database, we are going to have three different relationships, which I will explain in this segment.

**One-to-Many:** The first relationship that I will explain is the One-to-Many relation. Imagine you have Movie entity and a Director entity, the relation between them is; one Director can have many Movies *(so a One-to-Many relation)*.

To code that so the entity framework understands it, one must first go over to the Movie class and add these lines;

```
public int DirectorID { get; set; }
public virtual Director Director { get; set; }
```

By making a variable and defining the type as `Director`, the framework will understand that the movie class want to create a relation to the director class. This way, the DirectorID becomes a foreign key in the Movie table, which means when you create a movie you can refer the movie to a director *(The int directorID variable is not necessary, but if you don't have it, the framework will give you a default name to the foreign key).*

Additionally, in the Director class you must add;

```
public virtual ICollection<Movie> Movies { get; set; }
```

By doing these steps, you will have "One-to-Many relation" in the database, making it possible that a Director can have more than one Movie element.

**One-to-One:** To create a One-to-One relationship the primary key acts as a foreign key and there is no separate foreign key column for either tables, like in our MyMovieList class and Rating class.

That means, in our MyMovieList class we have;

```
public int MyMovieListID { get; set; }
public virtual Rating Ratng { get; set; }
```

And in our Rating class we must have a primary key which also is a foreign key;

```
public int RatingID { get; set; }
public virtual MyMovieList MyMovieList { get; set; }
```

By doing it this way, the Entity Framework understands that you want a One-to-One relation.

**Many-to-Many:** To create a Many-to-Many relation you are going to need 3 classes. To give an example from our system we're going to talk about Movie and Genre, because the relation between them is a Many-to-Many.

First, you need to add this line of code to the movie and genre class;

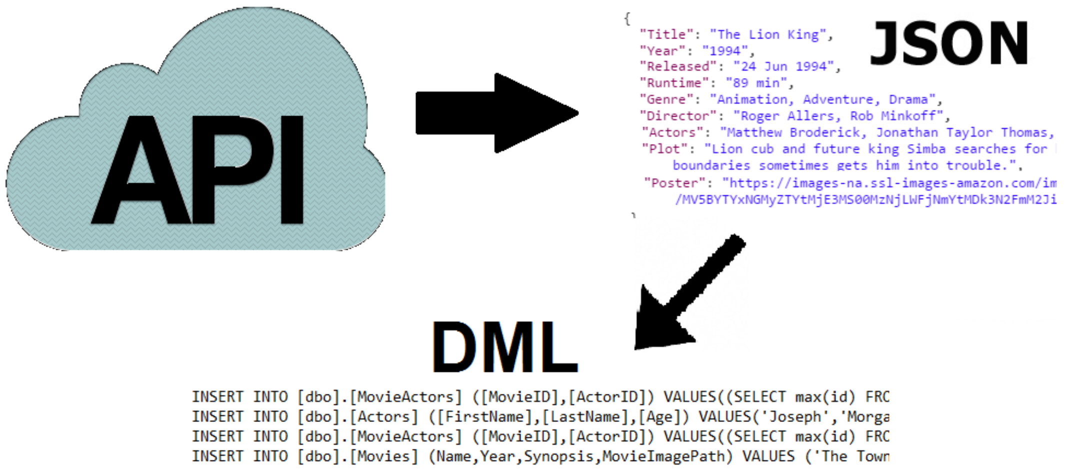
```
        public virtual ICollection<MovieGenre> MoviesGenres{ get; set; }
```

Then you make a class called MovieGenre and you add these variables

```
        public int MovieID { get; set; }
        public virtual Movie Movie { get; set; }

        public int GenreID { get; set; }
        public virtual Genre Genre { get; set; }
```

By doing these steps and updating the database from the console, the framework will understand that you have made a Many-to-Many relation and therefore the MovieID and GenreID will act as foreign keys.

## Data Entry *- (Casper)*

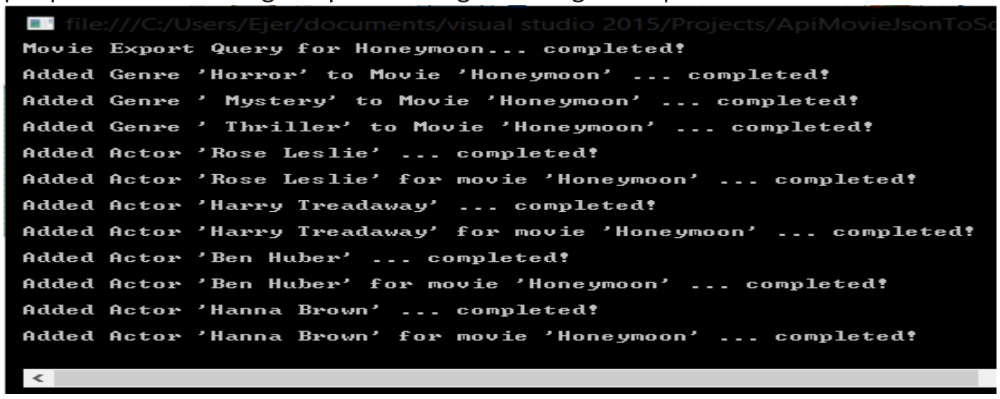


```
{
  "Title": "The Lion King",
  "Year": "1994",
  "Released": "24 Jun 1994",
  "Runtime": "89 min",
  "Genre": "Animation, Adventure, Drama",
  "Director": "Roger Allers, Rob Minkoff",
  "Actors": "Matthew Broderick, Jonathan Taylor Thomas,
  "Plot": "Lion cub and future king Simba searches for
      boundaries sometimes gets him into trouble.",
  "Poster": "https://images-na.ssl-images-amazon.com/im
      /MV5BYTYxNGMyZTYtMjE3MS00MzNjLWFjNmYtMDk3N2FmM2Ji
}
```

```
INSERT INTO [dbo].[MovieActors] ([MovieID],[ActorID]) VALUES((SELECT max(id) FRC
INSERT INTO [dbo].[Actors] ([FirstName],[LastName],[Age]) VALUES('Joseph','Morga
INSERT INTO [dbo].[MovieActors] ([MovieID],[ActorID]) VALUES((SELECT max(id) FRC
INSERT INTO [dbo].[Movies] (Name,Year,Synopsis,MovieImagePath) VALUES ('The Town
```

We thought that randomly generated data, wasn't the best way to present and populate our database with. Therefore, we decided to create our own program to populate the tables with significant amount of realistic test data. We used the online API http://www.omdbapi.com/, which is a free web service to obtain movie information and movie images.

We have created a little program in C#, that basically send a request to the API with a given movie title, which returns a json-file with all the movie information we require. After we have gathered all the information we need, the program generates and export a DML query out of the json information we have obtained. The program will transform the data, we have obtained, with filter and regex expressions, so that the data type will match the ones in our database tables. We have written a long list of 200 movie titles, that the program should execute DML queries for. The program also find the movie genres and actors, and generate it to DML queries. A movie can have multiple genres and actors. but our program handles it by using a foreach loop. The program created total **2342** DML queries is out of 200 movie titles. The program also has a console window, to keep people informed along the process of generating DML queries.



```
file:///C:/Users/Ejer/documents/visual studio 2015/Projects/ApiMovieJsonToS
Movie Export Query for Honeymoon... completed!
Added Genre 'Horror' to Movie 'Honeymoon' ... completed!
Added Genre ' Mystery' to Movie 'Honeymoon' ... completed!
Added Genre ' Thriller' to Movie 'Honeymoon' ... completed!
Added Actor 'Rose Leslie' ... completed!
Added Actor 'Rose Leslie' for movie 'Honeymoon' ... completed!
Added Actor 'Harry Treadaway' ... completed!
Added Actor 'Harry Treadaway' for movie 'Honeymoon' ... completed!
Added Actor 'Ben Huber' ... completed!
Added Actor 'Ben Huber' for movie 'Honeymoon' ... completed!
Added Actor 'Hanna Brown' ... completed!
Added Actor 'Hanna Brown' for movie 'Honeymoon' ... completed!
```

You can see the source our program at the attached appendix “DML Program” pdf file

**<u>Data Manipulation Language</u>:**
The DML queries are too long for our report, therefore we have attached the DML log to the appendix “DML Queries “pdf file.

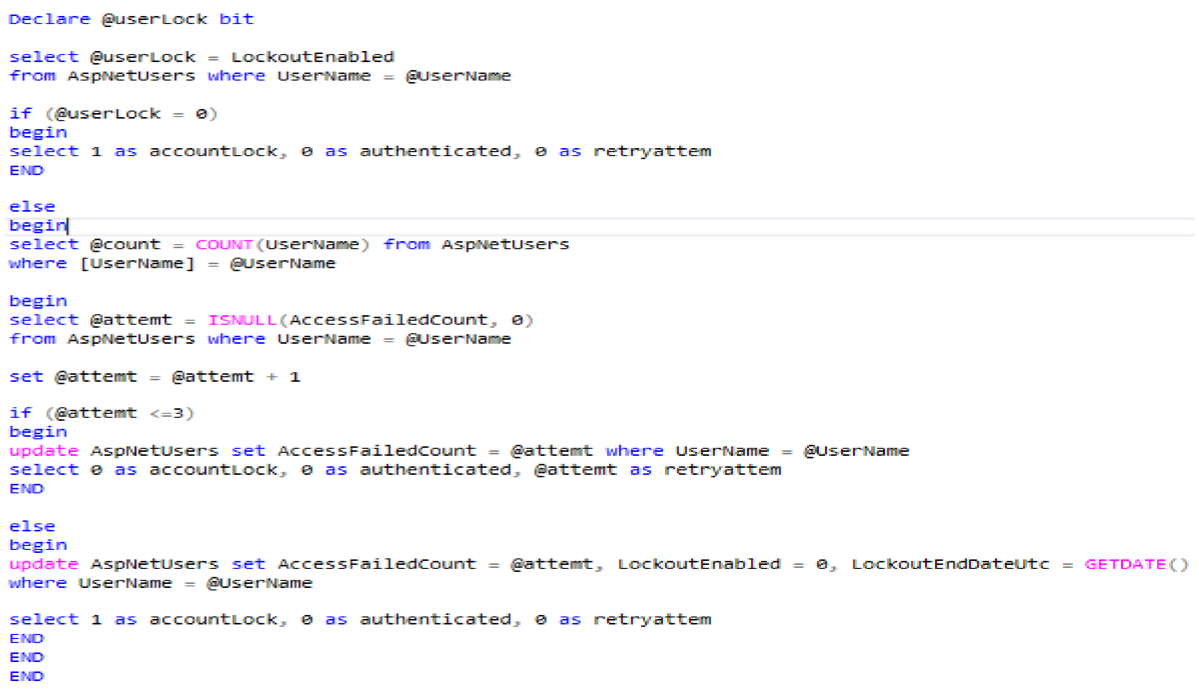
## Stored procedure *- (Benyam)*

Stored procedure is like a programming function for database you can use to perform a query. Outthought there are many cons for why to use a stored procedure, well to start with some of the advantage for using stored procedure are maintaining, security, usability and performance. We haven't been able to fully check and work with it so that we could experiment and see the benefit of stored procedure.  If we take maintainability for example, a developer has to either work with a project that involve a database that is used by many different applications hence maintainability would be easier if some changes happen. This been said, but in the end the cons and understanding we got from using stored procedure are comprehensive.

We used stored procedure for our login function, we needed to check if users had provided a correct user name and password to login on our website. If we were going to validate the user, the only true query we had to check if they exist and if they don't we would lock the user and update the user table with relevant data. This would result that we had to check both in the application and database, hence the traffic between the application and the database would be more than what we have now hence lower performance.

*How we used it*

The stored procedure we created take user parameters and execute a function that check if user is in the database, if he is it return user and if user is in the database but have locked it, return user is locked for login. Last if user is in the database but have given wrong password the function will update the database for login attempts if he tries more than three times, the function will lock the user and update the database with date and the user will be locked for future logins.



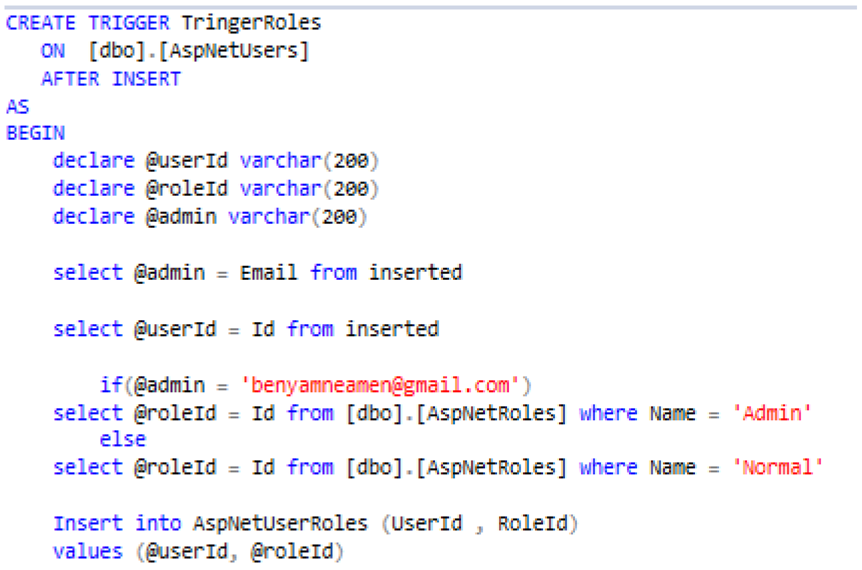
```
Declare @userLock bit

select @userLock = LockoutEnabled
from AspNetUsers where UserName = @UserName

if (@userLock = 0)
begin
select 1 as accountLock, 0 as authenticated, 0 as retryattem
END

else
begin
select @count = COUNT(UserName) from AspNetUsers
where [UserName] = @UserName

begin
select @attemt = ISNULL(AccessFailedCount, 0)
from AspNetUsers where UserName = @UserName

set @attemt = @attemt + 1

if (@attemt <=3)
begin
update AspNetUsers set AccessFailedCount = @attemt where UserName = @UserName
select 0 as accountLock, 0 as authenticated, @attemt as retryattem
END

else
begin
update AspNetUsers set AccessFailedCount = @attemt, LockoutEnabled = 0, LockoutEndDateUtc = GETDATE()
where UserName = @UserName

select 1 as accountLock, 0 as authenticated, 0 as retryattem
END
END
END
```

The good part of this is let's say as an example we had to develop a new website that require the same type of user login functionality, this would make our work easier because all we had to do was create this stored procedure on the new database and test it with the new requirement and then just implement the business logic on the new application for this to work.

## Triggers *- (Benyam)*

The most benefit for using trigger is if after insert, update or delete you need to modify two or more tables with data trigger, it comes really handy. The down side is that triggers are hidden and happen in the background this would make them really hard to detect especially if many people are working with the same project.

We wanted to assign a user to a role when they register the first time, if we would do this in the application first we would register the user than take the user Id and get the role Id and assign those two value to the user Role table. What we have now with trigger is that in our application we only write the code to register the user, but the database fire a trigger every time a user register and assign them to the role in the background.



```
CREATE TRIGGER TringerRoles
   ON  [dbo].[AspNetUsers]
   AFTER INSERT
AS
BEGIN
    declare @userId varchar(200)
    declare @roleId varchar(200)
    declare @admin varchar(200)

    select @admin = Email from inserted

    select @userId = Id from inserted

        if(@admin = 'benyamneamen@gmail.com')
    select @roleId = Id from [dbo].[AspNetRoles] where Name = 'Admin'
        else
    select @roleId = Id from [dbo].[AspNetRoles] where Name = 'Normal'

    Insert into AspNetUserRoles (UserId , RoleId)
    values (@userId, @roleId)
```

The trigger start with declaring the variable that was going to be inserted into the database, after that the function assign the value from inserted and with a select statement from database table. Once we have the data it's just to insert the values to the database table in our case AspNetUserRoles.

## Transactions - (Casper)

A transaction is a group of database commands that are treated as single unit
Before our transactions is fully successful, they must pass the "ACID" test, that is:

**Atomic** - All statements in the transaction either completed successfully or they were all rolled back. The task that set of operations presents is either accomplished or not, but in any case not left half-done.

**Consistent** - All data touched by the transaction is left in a logically consistent state.

**Isolated** - The transaction must affect data without interfering with other concurrent transaction, or being interfered with by them. This prevents transactions from making changes to data based on uncommitted information.most databases use locking to maintain transaction isolation.

**Durable** - Once a change is made, it is permanent. If a system error / power failure occurs before a set of commands is complete, then those commands are undone and the data is restored to its original state once the system runs normal again.

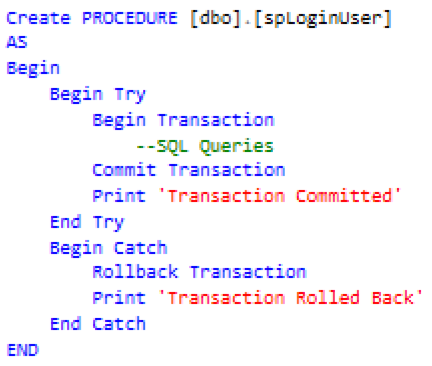
All our transactions have passed the “ACID” test.

## Transaction Process Flow - (Casper)

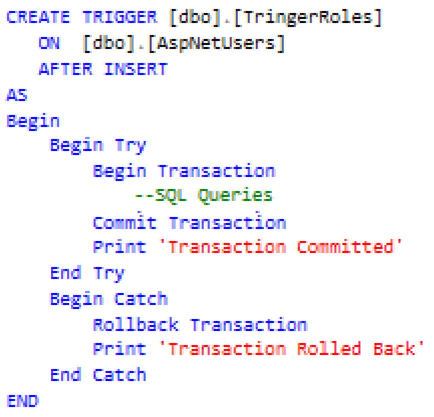
1. Begin Transaction
2. Process Database Commands
3. Check for Errors

a. if errors occurred
    **Rollback the Transaction**
  b. else
    **Commit the Transaction**

This is the process flow for our transactions.
We start by beginning the transaction and process the database commands. Then we check the commands for errors to occur. If the commands have no errors the transaction will be committed, otherwise if the commands do contain error(s) the transaction will undo itself, and roll back to its original state.

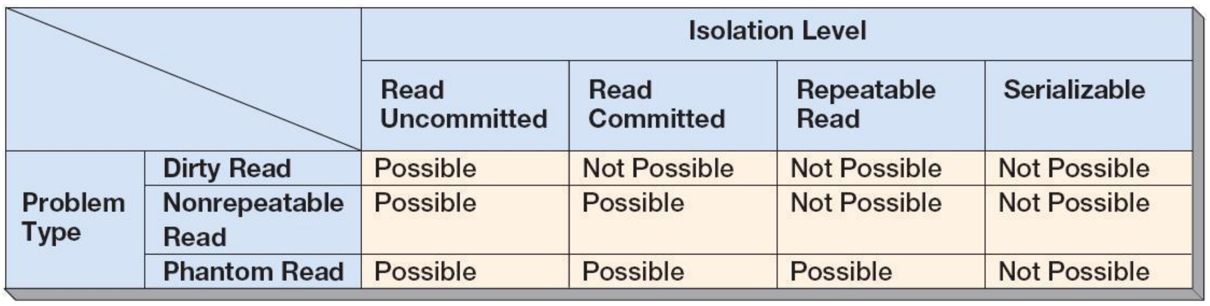


```
Create PROCEDURE [dbo].[spLoginUser]
AS
Begin
    Begin Try
        Begin Transaction
            --SQL Queries
        Commit Transaction
        Print 'Transaction Committed'
    End Try
    Begin Catch
        Rollback Transaction
        Print 'Transaction Rolled Back'
    End Catch
END
```



```
CREATE TRIGGER [dbo].[TringerRoles]
   ON  [dbo].[AspNetUsers]
   AFTER INSERT
AS
Begin
    Begin Try
        Begin Transaction
            --SQL Queries
        Commit Transaction
        Print 'Transaction Committed'
    End Try
    Begin Catch
        Rollback Transaction
        Print 'Transaction Rolled Back'
    End Catch
END
```

## Isolation Levels - (Casper)



| | | Isolation Level | | | |
|---|---|---|---|---|---|
| | | Read Uncommitted | Read Committed | Repeatable Read | Serializable |
| Problem Type | Dirty Read | Possible | Not Possible | Not Possible | Not Possible |
| | Nonrepeatable Read | Possible | Possible | Not Possible | Not Possible |
| | Phantom Read | Possible | Possible | Possible | Not Possible |

Our default isolation level for our database server have been set to level 2 READ COMMITTED. This level prevents dirty reads, because the statements cannot read data that has been modified but not committed, by other transactions. Non-Repeatable- and phantom reads is still possible to occur at this level. A non-repeatable read is one transaction attempts to access the same data twice and a second transaction modifies the data between the first transaction's read attempts. We can prevent this and avoid this problem, by making the read transaction lock the value of the data so it doesn't change, until the read transaction is completed. The write transaction can commit only after the read commits. Phantom read happens when one transaction executes a query twice and it gets different number of rows in the result set each time. This might occur when a second transaction insert a new row that matches the WHERE clause of the query executed by the first transaction. To prevent this read problem, we could set the transaction isolation level to level 1 Serializable. The level will place a range lock on the rows between 1 and 3, which prevents any other transaction form inserting new rows within that range.

## Transaction Management - (Casper)

A deadlock can occur when two processes hold locks on a page on which the other process holds a conflicting lock. Even so deadlocks cannot be completely avoided, we can still minimize the possibility for a deadlock to occur. We can minimize the change by following a certain coding convention:

- **Access objects in the same order**

Have all transactions access objects in the same order, which makes deadlocks less likely to occur.

- **Avoid User Interaction in Transactions**

Transactions that include user interaction are more likely to block other transactions. Imagine a situation where a transaction is waiting for a user input for its parameters, and then the user goes for lunch. This would result in others transactions being blocked while waiting for the first transaction to complete, because any locks hold by the transaction are only released when the transaction is committed or rolled back.

- **Avoid long running transactions**

Its best to make the transactions smallest as possible and make it commit as soonest as possible. Because data might change over time.

- **Use a Lower Isolation Level**

We can minimize the chance of locking contention, by determine whether a transaction can run at a lower isolation level. In some cases, we can even use row versioning rather than shared locks during read operations.

## Concurrency Control - (Casper)

Concurrency control is when modifications made by different users that may affect each other in a concurrency. Concurrency control can be broadly divided into two categories:

- **Pessimistic concurrency control**

Locks data that has been read and stays under preparation for update, so no one can use it at the same time.

- **Optimistic concurrency control**

Does not lock the data, but makes a check after the update, if the original data has been changed during the transaction, an error occurs and transaction rolls back.

## Queries - Real-life operations for users - (Mert)

We will be running some different queries in this part, where we will simulate real-life operations that users are able to do, such as searching for a movie.
Our idea is to make use of different types of queries, and explain what the difference between them is and what you could benefit from using one from another.

In MovieBasen you have the option to search for all the movies that match with a specific pattern in sql;

```
SELECT Movies.Name, Movies.Year FROM Movies WHERE Name LIKE '%The%'
```

or search it directly;

```
SELECT Movies.Name, Movies.Year FROM Movies WHERE Name = 'The 13th Warrior'
```

If we are going to find all the movies that a single actor has been involved in, we will have to make a join;

```
SELECT dbo.Movies.Name, dbo.Movies.Year FROM dbo.MovieActors
JOIN dbo.Movies ON (Movies.ID = MovieActors.MovieID)
WHERE ActorID = 38
```

We will also be able to see the genre and actor for a specific movie that we choose;

```
SELECT DISTINCT m.Name, a.FirstName, a.LastName, g.Name
FROM dbo.Actors AS a
INNER JOIN dbo.MovieActors AS ma ON ma.ActorID = a.ID
INNER JOIN dbo.Movies AS m ON m.ID = ma.MovieID
INNER JOIN dbo.MovieGenres AS mg ON mg.MovieID = m.ID
INNER JOIN dbo.Genres AS g ON g.ID = mg.GenreID
WHERE m.Name = 'Saw'
```

With this we will list all the actors and genres that have been added to the movie. We are using inner join which returns all rows when there is at least one match in both tables.

With another sql statement we will look for all the movies between a year interval;

```
select dbo.Movies.Name, dbo.Movies.Year from dbo.Movies where Year like '200_'
```

Here in this case we will get all the movies that is between 2000 to 2009.

### Data Optimization: How do we optimize queries? - *(Mert)*

We will i look at how we can optimize our queries, the best way, by for instance looking at our execution plan for the different sql statements.

Some things to look for when optimizing the queries are for instance fat pipes/thick lines which is the volume of how many rows that is being moved. If a skinny line suddenly becomes a thick line or opposite when running a query, it will most likely be an indication that something is wrong.  Or if there’s too many unnecessary operator, it will also slow the query down.

When optimizing queries, we are going to have different things in focus, for instance how much we are selecting or which joins and types of sql statements that is executed. Instead of making a select of all (*), we will only select the attributes that is necessary.

Another important thing to do, in order to optimize your sql queries is the number of rows that is going to be read. The fewer the rows there are to be read, the faster will the query run.

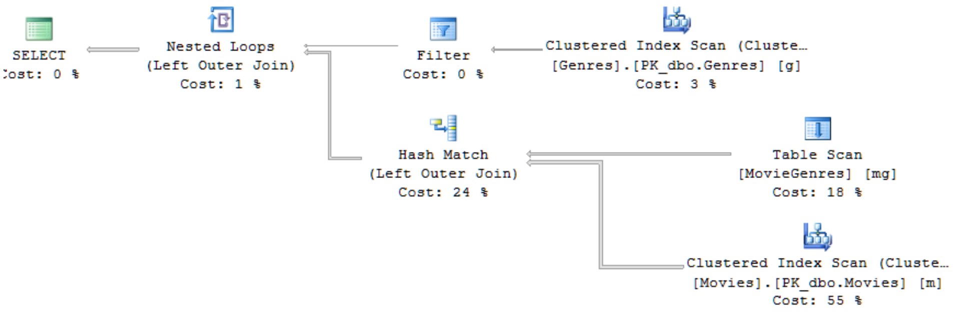
In the execution plan, we will have to look at the details about estimated operator cost, estimated I/O cost, estimated CPU cost and (actual) number of rows read.
Estimated CPU cost is how much impact the query have on the CPU, the estimated I/O cost how much resource (Input/output) the query is using. Estimated operator cost, is the percentage cost taken by the operator.

After running the query, we will look at the execution plan to see what happens inside the query.

We will look at two different join methods (inner join and full outer join), where we will join 2 tables in order to get all the movies in a specific category to see which one is faster to use in this case;

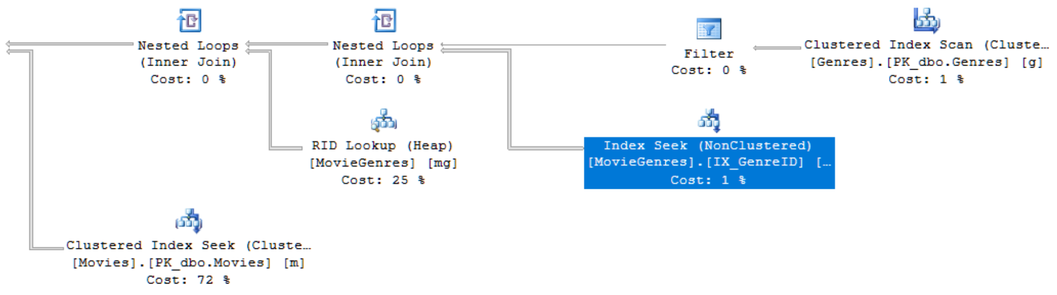
```
SELECT Movies.Name, dbo.Genres.Name FROM dbo.Genres
FULL OUTER JOIN dbo.MovieGenres ON MovieGenres.GenreID = Genres.ID
FULL OUTER JOIN dbo.Movies ON Movies.ID = MovieGenres.MovieID
WHERE dbo.Genres.Name = 'Horror'

SELECT Movies.Name, dbo.Genres.Name FROM dbo.Genres
INNER JOIN dbo.MovieGenres ON MovieGenres.GenreID = Genres.ID
INNER JOIN dbo.Movies ON Movies.ID = MovieGenres.MovieID
WHERE dbo.Genres.Name = 'Horror'
```

**Full Outer Join:**



In the full outer join, there's a clustered index scan for Movies table, and a table scan for MovieGenres (retrieves all rows from table) those two have a cost at 55 (0,06802) and 18 (0,02224) percent. Afterwards it gets into a hash match, which is where the hash join first reads one of the inputs and hashes the join column where it later puts the resulting hash and the column values in a hash table which is in the memory.
There's also a clustered index scan for the Genres table, which also goes through a nested loop just like the hash match and then the values are returned.

**Inner Join:**


Nested Loops
(Inner Join)
Cost: 0 %
Nested Loops
(Inner Join)
Cost: 0 %
Filter
Cost: 0 %
Clustered Index Scan (Cluste…
[Genres].[PK_dbo.Genres] [g]
Cost: 1 %
RID Lookup (Heap)
[MovieGenres] [mg]
Cost: 25 %
Index Seek (NonClustered)
[MovieGenres].[IX_GenreID] […
Cost: 1 %
Clustered Index Seek (Cluste…
[Movies].[PK_dbo.Movies] [m]
Cost: 72 %

If we take a look at the execution plan for the inner join, the clustered index scan for the Genres table is the same. There's a difference with the MovieGenres, where there's a non-clustered index seek, to scan a specific range of rows from a non-clustered index.
As there isn't any clustered index for MovieGenres the RID Lookup is used, which is a lookup into a heap table with the row id.
There's also a clustered index scan for the Movies table, in the last part, which have the most cost compared to the other scans/operations.

In these two different parts in the execution plan we will compare some of the parts. We can for instance see how many number of rows that is read (from MovieGenres table), here we can see that in the full outer join (the left) there's read 4448 rows (from 216 rows) compared to the inner join (the right) we can see that there's only been read 216 rows.
We can also see that the estimated operator cost in the full outer join is much higher than in the query with inner join.

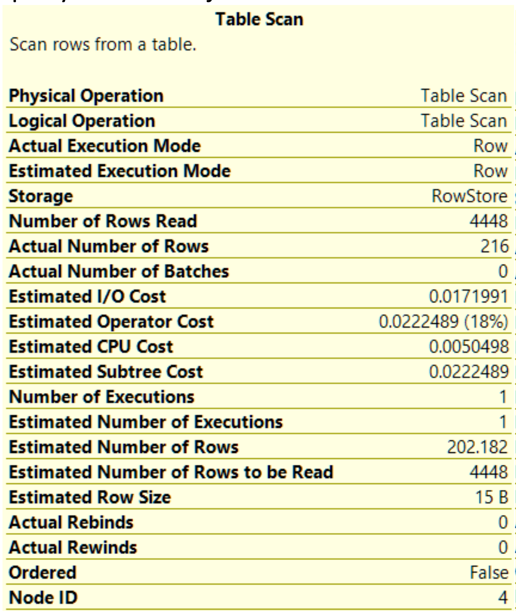
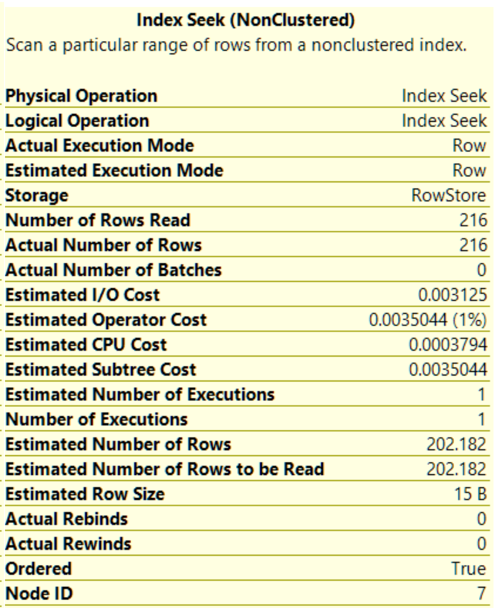


**Table Scan**

Scan rows from a table.

| | |
|---|---|
| **Physical Operation** | Table Scan |
| **Logical Operation** | Table Scan |
| **Actual Execution Mode** | Row |
| **Estimated Execution Mode** | Row |
| **Storage** | RowStore |
| **Number of Rows Read** | 4448 |
| **Actual Number of Rows** | 216 |
| **Actual Number of Batches** | 0 |
| **Estimated I/O Cost** | 0.0171991 |
| **Estimated Operator Cost** | 0.0222489 (18%) |
| **Estimated CPU Cost** | 0.0050498 |
| **Estimated Subtree Cost** | 0.0222489 |
| **Number of Executions** | 1 |
| **Estimated Number of Executions** | 1 |
| **Estimated Number of Rows** | 202.182 |
| **Estimated Number of Rows to be Read** | 4448 |
| **Estimated Row Size** | 15 B |
| **Actual Rebinds** | 0 |
| **Actual Rewinds** | 0 |
| **Ordered** | False |
| **Node ID** | 4 |



**Index Seek (NonClustered)**

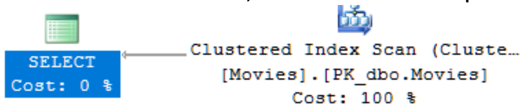
Scan a particular range of rows from a nonclustered index.

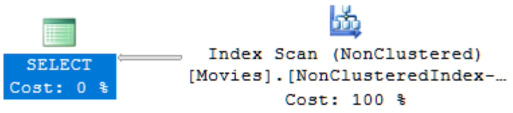
| | |
|---|---|
| **Physical Operation** | Index Seek |
| **Logical Operation** | Index Seek |
| **Actual Execution Mode** | Row |
| **Estimated Execution Mode** | Row |
| **Storage** | RowStore |
| **Number of Rows Read** | 216 |
| **Actual Number of Rows** | 216 |
| **Actual Number of Batches** | 0 |
| **Estimated I/O Cost** | 0.003125 |
| **Estimated Operator Cost** | 0.0035044 (1%) |
| **Estimated CPU Cost** | 0.0003794 |
| **Estimated Subtree Cost** | 0.0035044 |
| **Estimated Number of Executions** | 1 |
| **Number of Executions** | 1 |
| **Estimated Number of Rows** | 202.182 |
| **Estimated Number of Rows to be Read** | 202.182 |
| **Estimated Row Size** | 15 B |
| **Actual Rebinds** | 0 |
| **Actual Rewinds** | 0 |
| **Ordered** | True |
| **Node ID** | 7 |

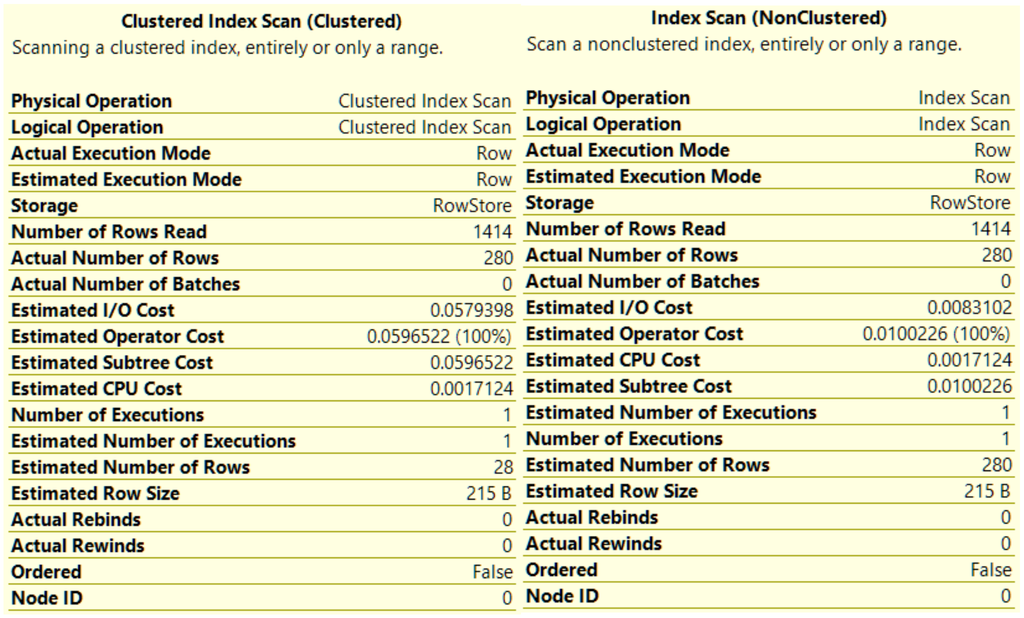
Overall we can conclude that for the most part, that the full outer join would have a higher cost than the inner join, mostly because it's reading much more rows than necessary. For instance, in the Movies table, the full outer joins read all rows (1616), and the inner join only reads 216. The inner join will generally have a lower estimated operator and CPU cost, which makes sense. Therefor will be the best query to use in this particular case.

**Search Optimizing**

Another thing we will look at is the speed and cost at how much it takes to search all movies for a specific year (2016). First we are going to do it with a clustered index scan, and next with non-clustered index scan, we will then compare those two.







**Clustered Index Scan (Clustered)**
Scanning a clustered index, entirely or only a range.

| | |
|---|---|
| **Physical Operation** | Clustered Index Scan |
| **Logical Operation** | Clustered Index Scan |
| **Actual Execution Mode** | Row |
| **Estimated Execution Mode** | Row |
| **Storage** | RowStore |
| **Number of Rows Read** | 1414 |
| **Actual Number of Rows** | 280 |
| **Actual Number of Batches** | 0 |
| **Estimated I/O Cost** | 0.0579398 |
| **Estimated Operator Cost** | 0.0596522 (100%) |
| **Estimated Subtree Cost** | 0.0596522 |
| **Estimated CPU Cost** | 0.0017124 |
| **Number of Executions** | 1 |
| **Estimated Number of Executions** | 1 |
| **Estimated Number of Rows** | 28 |
| **Estimated Row Size** | 215 B |
| **Actual Rebinds** | 0 |
| **Actual Rewinds** | 0 |
| **Ordered** | False |
| **Node ID** | 0 |

**Index Scan (NonClustered)**
Scan a nonclustered index, entirely or only a range.

| | |
|---|---|
| **Physical Operation** | Index Scan |
| **Logical Operation** | Index Scan |
| **Actual Execution Mode** | Row |
| **Estimated Execution Mode** | Row |
| **Storage** | RowStore |
| **Number of Rows Read** | 1414 |
| **Actual Number of Rows** | 280 |
| **Actual Number of Batches** | 0 |
| **Estimated I/O Cost** | 0.0083102 |
| **Estimated Operator Cost** | 0.0100226 (100%) |
| **Estimated CPU Cost** | 0.0017124 |
| **Estimated Subtree Cost** | 0.0100226 |
| **Estimated Number of Executions** | 1 |
| **Number of Executions** | 1 |
| **Estimated Number of Rows** | 280 |
| **Estimated Row Size** | 215 B |
| **Actual Rebinds** | 0 |
| **Actual Rewinds** | 0 |
| **Ordered** | False |
| **Node ID** | 0 |

When looking at the result of the same sql statement, first with the clustered index scan then with a non-clustered index to the attributes. The number of rows read is the same, but the estimated CPU cost is different, where the clustered index has a slightly higher cost. The estimated operator cost is also almost 6 times more and the estimated I/O cost is also way higher than the non-clustered index scan.

After comparing the clustered and the non-clustered, we can conclude that non-clustered is much faster, and great to use if you want to optimize your queries.

## Security and Access rules *- (Benyam)*

To secure our database and application we took two different approaches, first we secured our database so that we made roles that we granted to ourselves. Beside the administrator role that could basically do everything from CRUD of table, users and roles to some staff that we don’t know about now, we had staff and superstaff roles and we made users login for all of us and assigned us to the roles. The reason we do this is to illustrate how in a company this would basically help for the integrity of the database, where you might have some developers that haven’t much experience and could basically drop a whole database table. The second reason was if an application get hacked from an outside/inside instead of getting access to all of the databases that are on server we just grant the application to a specific database.

The syntax for creating and assigning users and roles is straight forward where you specify the name and the role of the user you want to create, assign and grant them permission, we are using azure cloud services and the syntax is a little bit different like this example where we create role and add a user to a role;

**create role staff add sp_addrolemember mert**

**EXEC sp_addrolemember 'superStaff', 'casper';**

Securing the database does help if the users of the application have access to all the tables through the program and maybe could modify the data for this we made a role base login system where the only one that could write and alter database data was the administrator. We had the Admin and Normal user role that we assign to the users of the application.

## Reflection - (All)

When we started the database course we all had some kind of experience on how to implement databases for our applications, never did we thought there was more than that. What we have learned from the course and implementation of it in our project is knowledge that we are going to take with us to the real world.

From the requirements analysis to optimization, from designing the model to actual database, from normalization and optimization to function like transaction, stored procedure to name few are some of how we have learned to manage databases in an optimal way, last but not least database security.

The only downside we had was time in implementing our project, because of the lack of time there are some functions that we haven't implemented.

After creating the different queries for simulating the real-life operations for users, we had to do some optimizing of the queries.
We had lots of difficulties with understanding the execution plan, so we asked the teachers for some help for a better understanding. It truly improved learning how to optimize our sql queries, we for instance didn't know how to read the execution plan. But after understanding the meaning of the symbols and details, it got much easier to optimize the queries, where we made a search, first with clustered index scan and then with non-clustered index scan. We could afterwards see the improvement of the queries with the non-clustered index.

We think we have done a good job in implementing an optimal database for our project, we have calculated the cost, space, which query that served our purpose best. All in all, a fantastic journey where we have learned how to think, implement and manage a database for big companies.

We had a constructive and instructive progress in our project overall, where we learned more about the different aspects of database.

## Bibliography *- (All)*

Books:

- Database Systems a Practical Approach to Design, Implementation, and Management Fourth Edition - THOMAS M. CONNOLLY • CAROLYN E. BEGG UNIVERSITY OF PAISLEY

Web-pages:

- https://www.simple-talk.com/sql/performance/execution-plan-basics/
- https://msdn.microsoft.com/en-us/library/ms174377.aspx
- https://www.codeproject.com/articles/25600/triggers-sql-server
- https://msdn.microsoft.com/en-us/library/ms190782.aspx
- https://msdn.microsoft.com/en-us/library/ms378709(v=sql.110).aspx
- https://docs.oracle.com/cd/B19306_01/server.102/b14220/consist.htm