## Chosen database

*4.2.1. Database Type Choice*

By reviewing the database choices, we had for this project we considered the use of relational databases, non-relational databases, blockchain and data warehouse. Information provided stated that there are approximately 250 thousand volumes and 10 percent of them are out at loan at any given time, meaning, that close to 300,000 new entries would be created in one year just by lending books. However, it was determined that relational databases performance will not be seriously affected with this growth of information, as most commonly performed action: create is not affected by data size change and all other actions are expected to have their time to perform task increase no more than half a second in following 30 years (if the lending rate is not to increase). Furthermore, SQL has following benefits:

1. We have more experience working with relational databases (less time is wasted getting to know new technology).
2. Support for ACID.
3. Limitless indexing.
4. Relationships have constraints.
5. Data is structured and normalized.
6. Product is made for small number of users and not expected to have more than 4 concurrent users.

*4.2.2. Database Engines*

As for engines, there are several choices that we considered, for a relational database, some of which are: Oracle Database, SQL Server and MySQL; and since all three of them were using dialects of the same language (SQL), it went down to the very basics when we took the decision on which to use. As a final decision, we chose SQL Server 2017, because of the following: SQL Server executes and commits each instruction, unlike Oracle which requires explicit commands to commit the changes; its included in visual studio 2017; ease of use, since not only were we thought on how to use it, but also compared to Oracle, which gives so many other settings and configurations that can be set to the wrong value and effect the performance; and of course performance.

*4.2.3. System-Database*

**Technologies for data access:**

When deciding on which data access technology to use for development of the application, we came to the 3 most popular technologies on the internet right now:

1. ADO.NET
2. LINQ to SQL
3. ADO.NET Entity Framework

We chose to follow in the footsteps of the most popular products, as they most likely are the best solution and they have great support for them. Afterwards we obtained knowledge of these technologies and started comparing them. ADO.NET was one of our initial vision in creating the application, although it is easier to use in difficult scenarios, we determined, that its extensions - LINQ to SQL and Entity Framework are easier to use in casual scenarios and is faster to develop for and easier to maintain than ADO.NET. After reviewing and comparing the last two options we had, we concluded, that Entity Framework is easier to maintain and more powerful than LINQ to SQL, also as of the release of .NET 4.0, LINQ to SQL is often considered by many to be an obsolete framework.

When starting work in Entity Framework, we must decide which of the following methods to use for our project:



*Visual diagram used by the Framework to autogenerate the Database SQL script and the Data Model source code files*

**Model-First**

**Pros**:

1. Great when there is need to have a visualization of the database
2. Easy to use and understand when dealing with large data structures
3. Models can be updated accordingly, without data loss

**Cons**:

1. Autogenerated SQL scripts can lead to data loss in case of updates
2. Hard to have precise control over generated model classes



*SQL script used to create the Database and from that corresponding Data Model and diagram is generated*

**Database-First**

**Pros**:

1. Easy to use existing database for creating models
2. Change will be always performed on the database, so no data loss can occur

**Cons**:

1. Can be hard to update database when dealing with multiple instances
2. Less control over generated model classes, even more so than the Model-First approach



*Model classes from project used in Entity Framework generate the database accordingly to it*

**Code-First**

**Pros**:

1. No need for diagrams
2. Easy to maintain and develop code
3. Ideal for small-to-medium sized projects
4. Saves development time
5. Good control over generated database

**Cons**:

1. Requires knowledge of object-relational mapping
2. Maintaining the database can be tricky without suffering data loss (Mostly overcome by using Migrations, added in EF 4.3)

After reviewing options, we decided to go with the Database-First approach as we did not want to waste too much of our time on making the solution work, since we do not have a lot of time to begin with. Besides the already mentioned reason, this solution by many is considered one of the best for projects with main concern being development of database, we wanted to gain more experience in using this approach, have an easily maintainable code and there was no real need for diagrams. In the end the actual cons of Database-First were rendered useless, since they do not affect us at all, or are not the main concern of this task.