Georgia Tech Library

Dávid Szőke Bartosz Lachowicz Piotr Gzubicki Jakub Kida

University College of Northern Denmark

Top-up: Software Development Programme

Class:
PSU0220

Title:
Georgia Tech Library project

Project participants:

- Dávid Szőke
- Bartosz Lachowicz
- Piotr Gzubicki
- Jakub Kida

Supervisors:

- Gianna Belle
- Jens Henrik Vollesen

Submission date:
28.05.2020

Number of characters including white spaces, excluding cover page, table of contents and a list of references: 140 305

1. Preface

The motivation for this report was 1st semester of Software Development top-up program project. It is created by a team of four students. The report is a description of our achievements during the development cycle we conducted. The contents include the process of designing, optimizing, implementing, and securing the database. The purpose of this report is to document the steps we undertook to complete the given tasks.

2. Abstract

This report briefly describes the process of designing, optimizing, implementing, and securing the database which are part of the was 1st semester Software Development top-up program project. It contains a complete EER diagram created by our team and the documentation of our steps undertook to transform it into the database schema. Next section describes the process of the database normalization up to BCNF form. By examining the designed results, we documented the process of implementing the database. Furthermore, the actions to optimize the database were performed including non-cluster indexing. In the last section the security of the database is examined, and it includes documentation of the process of implementing the basic security practices ex. logins, users, and schemas.

General Problem: In lots of systems nowadays the database solutions are not suitable for current times. Security, performance and reliability might not be perfect. This is sometimes due to having an old system from times when the possibilities were limited. These systems should be updated to modern measures as attackers will also have modern tools and not having the solutions for today's times may make it easier for them to attack. Performance is also important as modern systems should work as fast as possible to satisfy the customers who have expectations of a flawless system.

Company: Georgia Tech Library is a library that has many books and many customers who are students and professors. It also has different levels of employees which requires different access levels. They need a new IT system where they can manage the library, see book catalogue, manage book loans. Their new system must have solutions that comply with modern requirements.

Problem statement: Georgia Tech Library needs a system where the IT solutions, including the database solutions are future proof, secure and performant. The company has many users and many transactions are going through them daily. The system made has to be able to withstand the load and ensure its users about a reliable and fast experience. The database of this system must have the best possible structure, security measures, access control and performant queries.

Problem definition: How we can create a database that will satisfy every standard?

- How to make sure that the database has a setup that makes it impossible for unauthorized users to reach data?
- How to make sure that everyone is only reaching data that their role in the system requires?
- How to make a structure that will hold the data in the best possible way and will make it possible to create performant queries?
- How to make sure that the queries have good performance and can use the resources of the database server the most efficient way?

Table of contents

3. Database

3.1 Introduction

During 1st semester of the top up program we have in depth learnt about Databases and its internal calculations. The focus of this program was to learn how, having provided business case communicate with stakeholders using common business languages and ER/EER Diagrams. Another issue was related with implementation of this database with respect to good practices such as Normal Forms. This project is going to validate our skills both in database implementation as well as requirements conversion using EER and EER conversion rules. We are going Our intentions is as well to optimize and secure our database. The goals are to create a working data storage starting from requirements analysis just as if it was an enterprise project.

3.2 Entities and relationships description

Having read the case of the project, it is necessary to create a plan of a database schema implementation. The description is detailed and allows to understand how the Georgia Tech Library works. Due to the time limitations, it is not possible to create the whole system in the scope of the project. Therefore, as a development team, we came to an agreement that we will implement a main feature that the library should provide which is borrowing and lending books from both librarian and student point of view. Nonetheless, a database schema was created for whole system, even though some of its part will not be provided in the release product. The reason for that is that the work on the project will be continued after the hand-in.

There are some entities that are clearly visible in the case description. It is possible to distinguish three kind of people. They are students, professors, and employees of the library. These three roles might overlap for one person. For instance, a student can also be a professor or a librarian. Personnel of the library can be categorized into chief librarian, departmental associate librarians, reference librarians, check-out staff and library assistants. In this case, one cannot have two roles at the same time. That means, it is not possible for assistant to be reference librarian. It essential to store basic information about a person like SSN, name, address, phone number, campus, and home address as well as time of how long certain person can borrow a book. This information is generic for all people. Nonetheless, there can be some more facts stored about students, like course name and a graduation date. Employees on the other hand have some unique data as number of weekly working hours, salary and hire date.

Despite user-related entities, there are also entities related with materials that are accessible in a library. Georgia Tech Library stores different kinds of books and maps. Depending on a condition of the book it might not be able to be borrowed. Besides that, maps as well as reference and rare books are not eligible for borrowing. Students can borrow books on their own or with the help of librarians. Each book is uniquely identified by its ISBN number. Additionally, each book contains information about its author, title, subject area, description, language, cover type, release date and number of copies. Books that can be lent also contains information of how much copies are available.

Each person identifies itself with a library card, while using a library. Card is issued by employees for a certain period, is has a unique number and a picture of its owner. The purpose of having a card is to borrow and return books.

Not only was it possible to identify these basic entities, but also to characterize relations between them. So far it is mentioned that librarians create cards for users. An employee can make many cards. For the sake of traceability, information about who created the card should be also recorded. The created card is associated only with one person.

The last main visible relationship is the act of borrowing books. This connection is between person and rental book entity. It is many-to-many relationship. It is because a person can borrow more than one book and a certain book will be borrowed multiple times by different people over the time.

Having the basic entities and their relationships detected, it is possible to create an enhanced entity-relationship diagram. We did it to plan a database more thoroughly by delving into the properties and constraints with more precision.

3.3 EER diagram

Because of the different responsibilities of the entities, they can be divided into three groups. Entities related with people, card, and library materials. Below, in three subsections each of the group is described.

3.3.1 Person-related entities

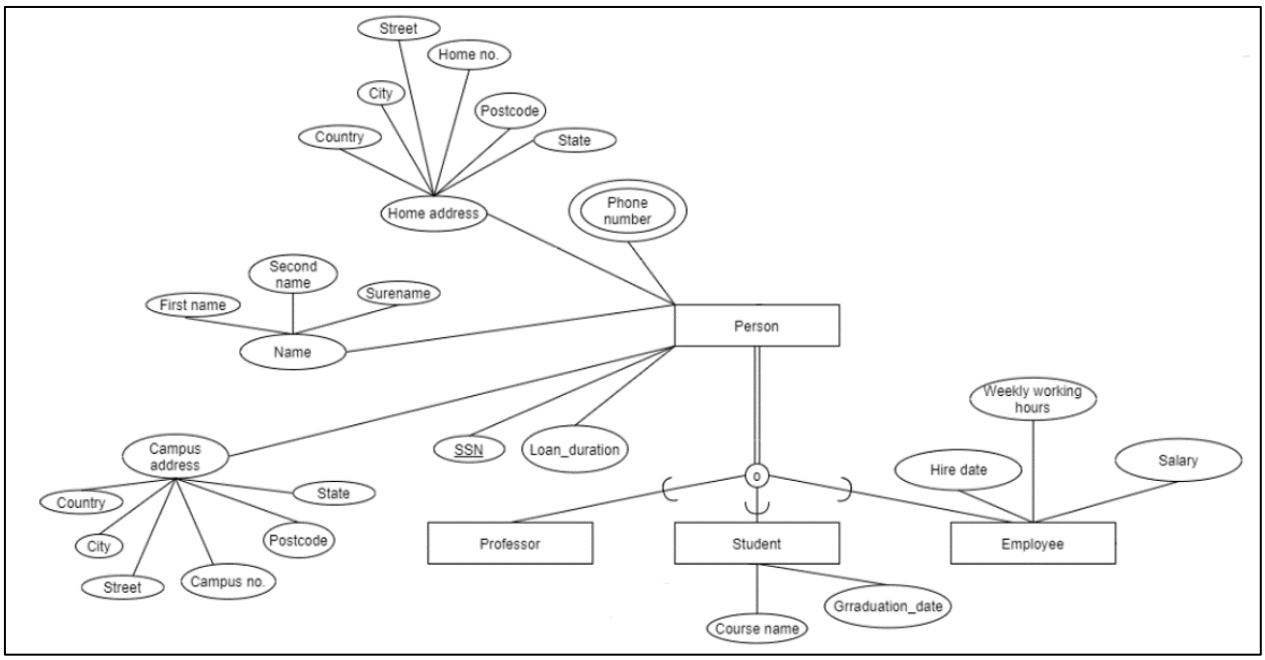


Figure 1 Person-related entities from EER diagram.

The above figure shows part of EER diagram where only person-related entities are present. As it is visible there is a Person entity. It consists of six attributes. The key attribute, that uniquely identifies each entity of Person is an SSN. The reason for that is because each person has a unique SSN number. Loan duration attribute determinates for how long a certain person can borrow books. There are also three composite attributes: name, campus address and home address. Last two attributes are the same. Nonetheless one of them determinates the address of a campus where a certain person is registered and the other one is an information about the person accommodation. Address information consists of information about country, city, street, campus number, postcode and a state. The third composed attribute is Name. It is made of first name, second name and surname attributes. Each person in our system must be either Professor, Student or Employee. That is why we have made a specialization. The specialization is overlapping, which means that a person might be for example both Student and Employee or Student and Professor. Student entity has three additional attributes, which are course name and graduation date. The Professor entity does not have any specified fields. However, an Employee entity contains additional: hire date, weekly working hours and salary attributes. The attributes for Student and Employee entities are specified only for those

entities. That is why they are not a part of Person entity but the child entities of a Person. Depending on the user that will be stored in the user, their attributes will differ.

3.3.2 Card entity

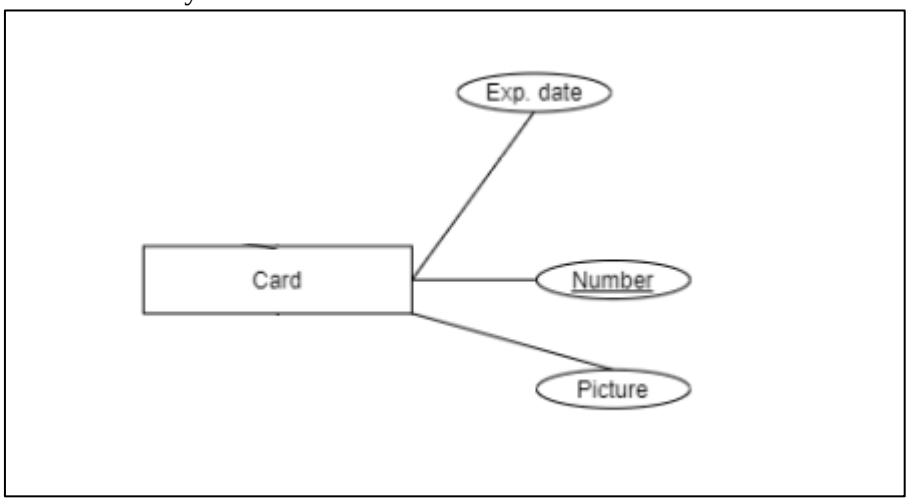


Figure 2 Card entity from EER diagram

Another entity that was defined from the description of the Georgia Tech Library is a Card entity. It is going to be used for identification of its owner to give them possibility of borrowing and returning books. For that reason, each card must have a unique number so that the users would not be confused with each other. The entity contains three attributes. They are number, expiration date and picture. Because of its uniqueness, the number attribute became a key.

3.3.3 Library materials related entities

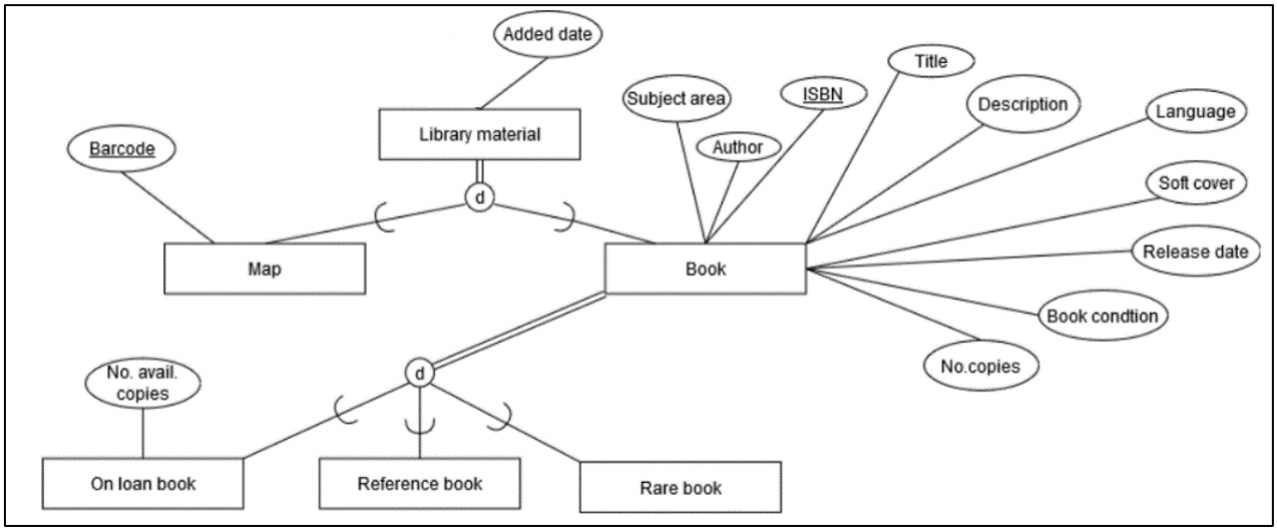


Figure 3 Library materials related entities from EER diagram.

The last entity group that was recognized is a group containing entities related with library materials. The two main ones are Map and Book entity. We decided to generalize them into a Library material entity. Despite, they have only one common attribute the justification of the generalization is the logic that Book, and Map are both part of materials that are present in a library. A library material can be either a book or a map but never two at the same time. That is why it is a disjoint constraint at the generalization. Map entity has only one attribute. It is a barcode which is a key. On the other hand, a book consists of multiple attributes.

They are subject area, author, ISBN, Title, Description, Language, Soft cover, Release date and number of copies. All of them are straightforward. ISBN is unique for a book, considering a book entity as not a copy of a specific book but rather as a specific publication. From the case description it can be deducted that a book entity can be specialized into on-loan book, reference book and rare book. The two last are not available for users to be borrowed. The one that users can lend is the on-loan book. Since these specializations cannot overlap, which means it is not possible for one book to be for example on-loan book and a reference book, there is a disjoint constraint present. On-loan book entity has an attribute which states how many copies are eligible to be borrowed.

3.3.4 Relationships in the GTL EER diagram

Not only does EER diagram consists of entities, their specializations, and generalizations but also relationships between them. As previously mentioned, there are three relations that we have defined while exanimating a Georgia Tech Library case. The first one is between Employee and Card entities.

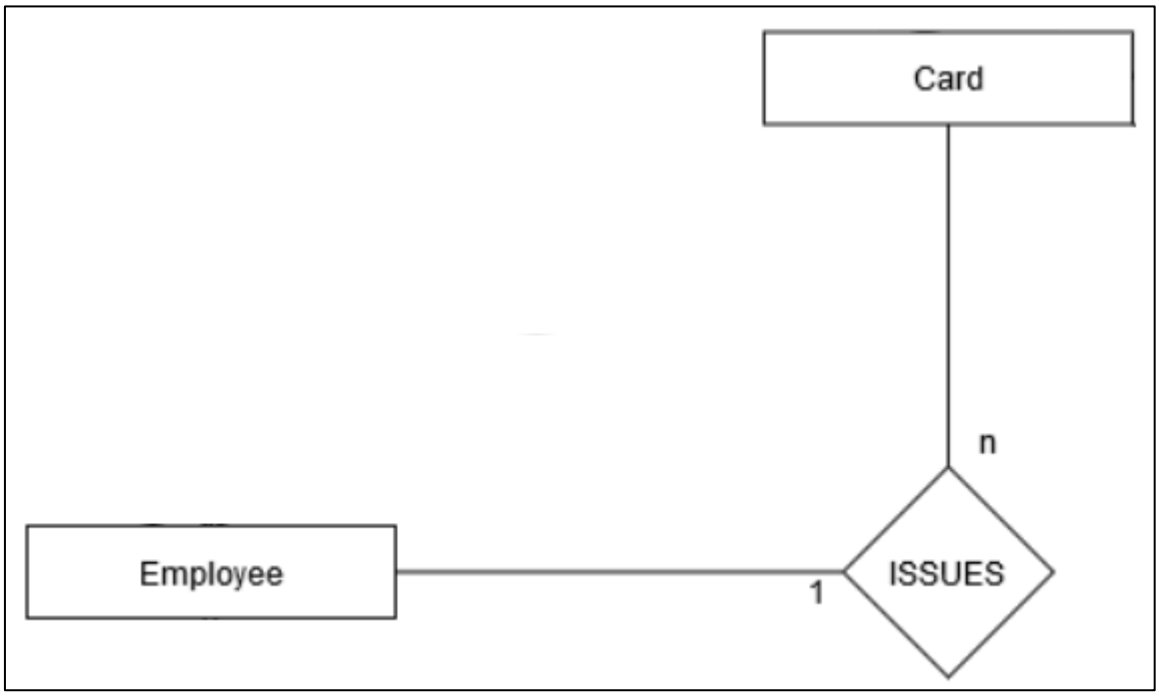


Figure 4 Employee-Card entities relationship.

The relationship is called ISSUES, the cardinality between card and employee is many to one. That means that a card is issued only by one employee and an employee might create multiple cards.

The next relationship is defined between the On Loan Book entity and the Person. It is a Person entity that has relationship with On Loan Book one, because everyone is capable of borrowing books. It does not matter if it is an employee, student, or professor. As soon as a certain person has a Card created, they can borrow book. The relationship points to On Loan Book entity, because it is the only child entity of a Book that can be lend. The relationship is many-to-many, which means that a book is borrowed by many different people over a time but not at the same period and a person can borrow more than one book. The relationship has also four attributes. They are return date, is_borrowed, borrow_date and payment. Payment is a calculated fee that a person is charged with if the return date extends available loan duration.

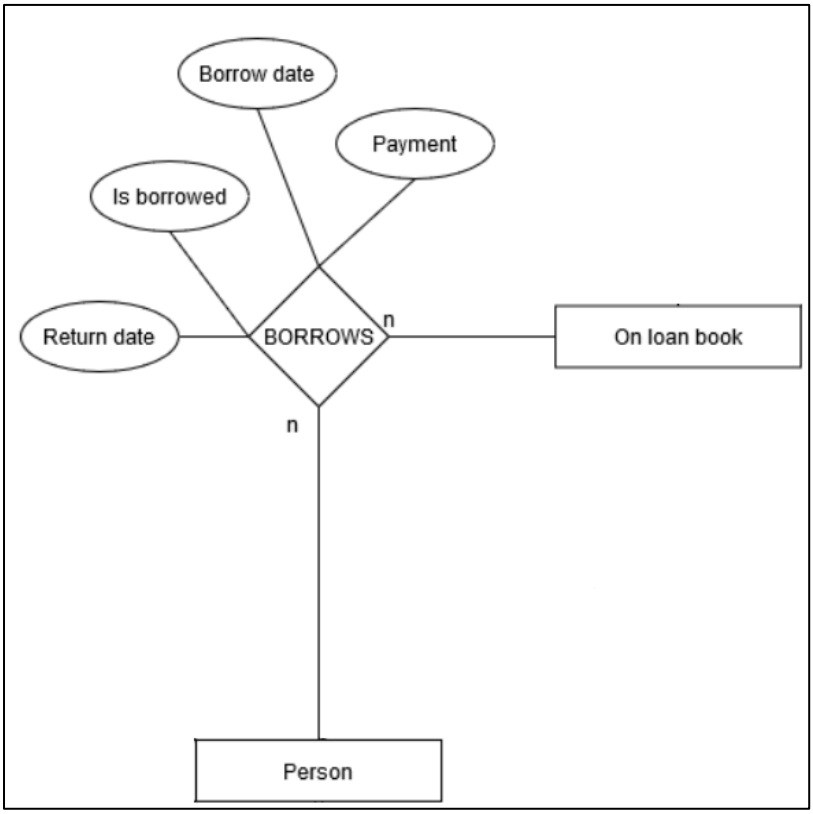


Figure 5 Person-"On loan book" relationship

The last relationship that we have included in our EER diagram is a relationship between Person and Card entity. It is required that the card entity must have a person associated with it, so that a relationship can exist. For that reason, a connection from card to the relationship is double lined (total participation). Moreover, it is one-to-one relationship. It means that a card can only have one person and vice versa.

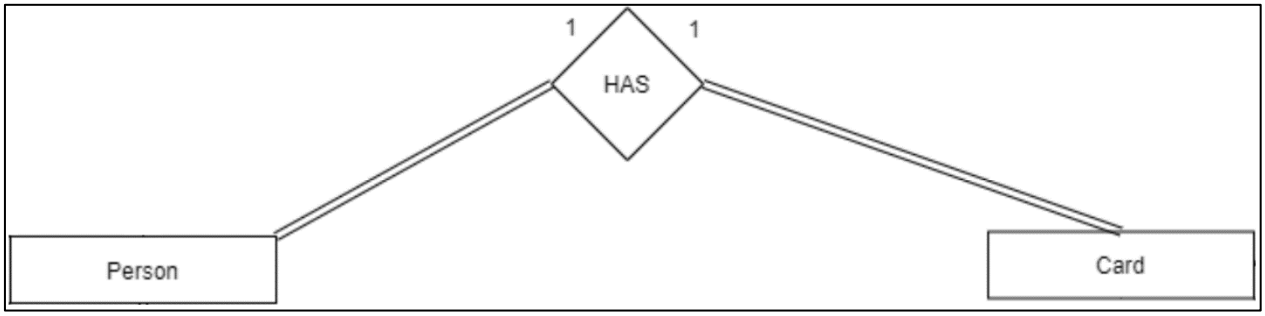


Figure 6 Person-Card relationship

3.3.5 Complete EER diagram

The complete EER diagram is displayed at the figure.

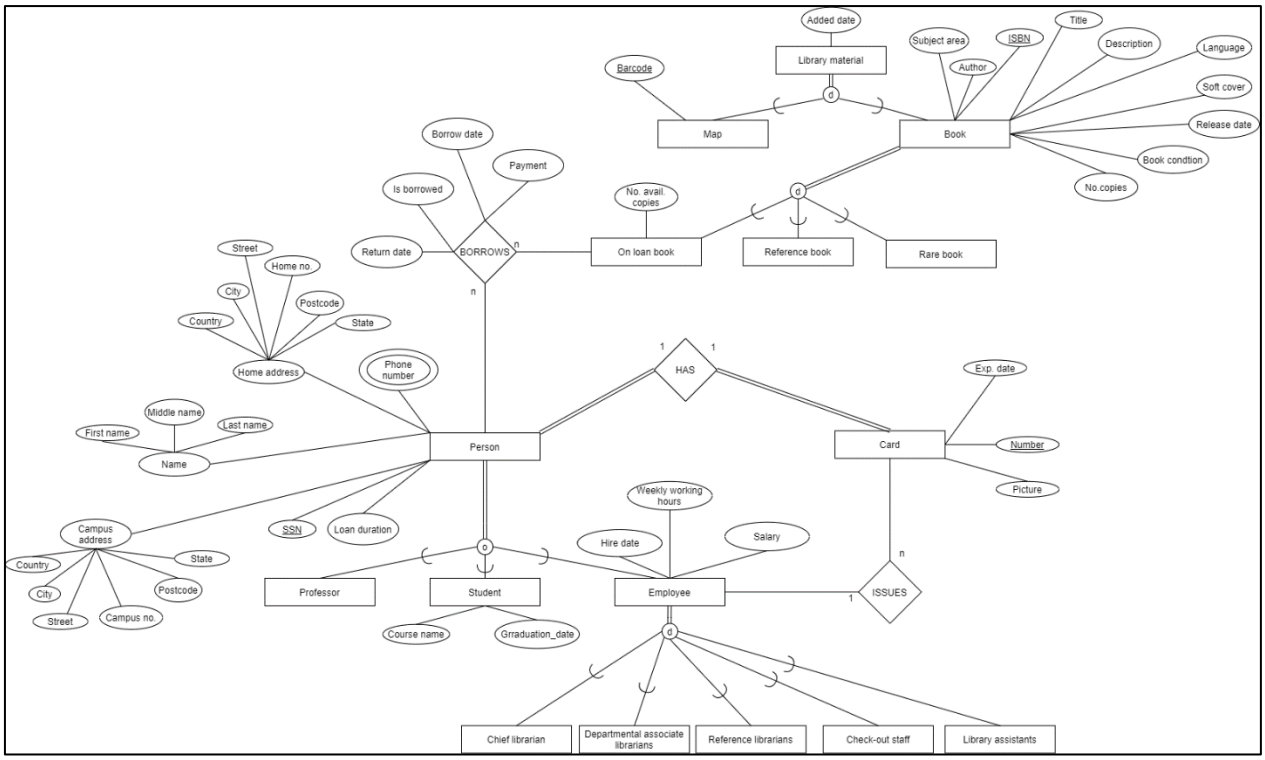


Figure 7 EER diagram of Georgia Tech Library

The next step of a database development is to transform the enhanced entity-relationship diagram into a database schema. It can be done by following pre-defined steps, which will be described in next section.

3.4 Transforming EER diagram into initial database schema

There are nine steps that need to be followed to transform EER diagram into a relational database. If we had only Entity Relational diagram, then we would have to follow only first seven steps. Last two are specifically dedicated for enhanced version of the diagram.

3.4.1 Step one: Mapping of Regular Entity Types

In this step, for each strong entity type in the diagram, it is required to create a relation that includes all simple attributes of the specific entity. While doing that, it is also essential to set a key of a relation. In our diagram we can see Person, Employee, Card, Map and Book entities. Each of them has simple key values. Consequently, they end up with the following schemas.

Person																
ssn	loan_duration	first_name	middle_name	last_name	home_country	home_city	home_street	home_address_number	home_postcode	home_state	campus_country	campus_city	campus_street	campus_address_number	campus_postcode	campus_state
Card																
Number	expiration_date	picture														
Book																
isbn	subject_area	author	title	description	language	cover	release_date	book_condition	copies_number							
Map																
barcode																

Figure 8 Database schema after first step of transforming of EER diagram

3.4.2 Step two: Mapping of weak entity types

In this step, for each weak entity type in the diagram, it is required to create a relation between the relation and its owner entity. At the same time include all simple attributes of a weak entity as attributes of a created relationship. In addition, include attributes of R as foreign key, the primary key attribute of the relation that points to the owner entity types. Thanks to that it is possible to identify the relationship type of a weak entity. Our diagram does not contain any weak entities, consequently there is nothing that can be transformed in the step.

3.4.3 Step three: Mapping of binary 1:1 relationship type

For the step it is needed to map 1:1 relationship in the EER diagram. To do that it is required to identify relations S and T that correspond to the entities having such a relation. There are three possibilities of archiving the right mapping. They are following:

1. Foreign key approach
   Choosing one relation and include as a foreign key in it the primary key from the other relation. The best practice is to choose a relation of an entity type with a total participation.
2. Merged relation approach
   The approach proposes to merge two entities, between which there is a 1:1 relationship, into a single relation. However, the approach is only possible when both participations are total.
3. Cross-reference or relationship relation approach.
   What the approach suggests establishing a third relation, that cross references the primary keys of the two relations representing the entity types. In result there would be created a lookup table. The table will contain tuples representing a relationship instances that relates corresponding tuples from the related schemas. The created relation includes primary key attributes of related schemas as foreign keys to them.

In our case there is only one 1:1 relationship. It is between Person and Card entities. We decided to follow the first approach. The person schema was appended with one more column. It contains a foreign key reference to a card schema's primary key constraint.

Consequently, the database schema looks as the following figure shows:

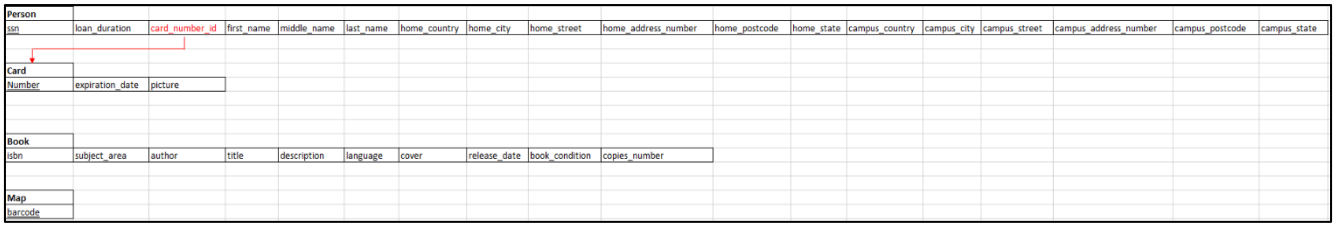


Figure 9 Database schema after third step of transforming of EER diagram

3.4.4 Step four: Mapping of binary 1:N relationship types

The next step is to map 1:N relationships. As previously it is possible to do that in more than one way. There are two approaches proposed.

1. The foreign key approach:

   The approach proposed to identify the relation that represents entity at the N side of the relationship side and include as a foreign key in it, the primary key of the other relation. It is possible, because each entity from N-side of a relation is related to at most once to the other entity instance.

2. The relationship relation approach:
   As an alternative, it is possible to use cross-references. It is the same approach as third option from 1:1 relationship. It demands creating a separate table with foreign keys relations to primary keys of related entities.

The EER diagram of a Georgia Tech Library has one 1:N relationship. It is between Employee and Card entities. The first entity is 1-side and the other is N-side of a relation. By using the foreign key approach, we modified card entity by adding columns containing foreign key reference to an Employee entity.

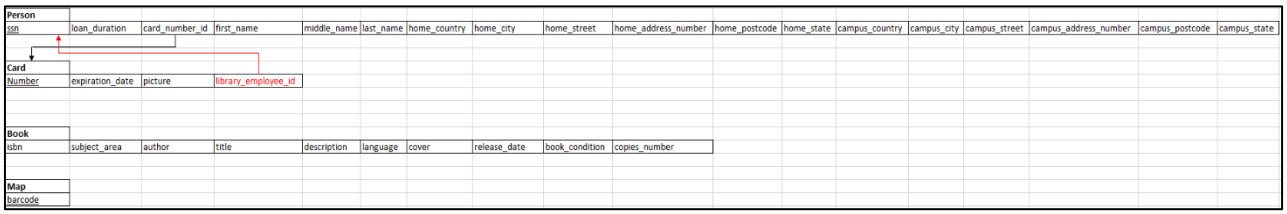


Figure 10 Database schema after forth step of transforming of EER diagram

3.4.5 Step five: Mapping of binary M:N relationship types

To map M:N relationships, in relational model with no multivalued attributes, the only option for M:N relationships is the relationship relation option. It was described in previous two steps. We did it by creating BookBorrow schema that consists of foreign keys relations to related schemas.

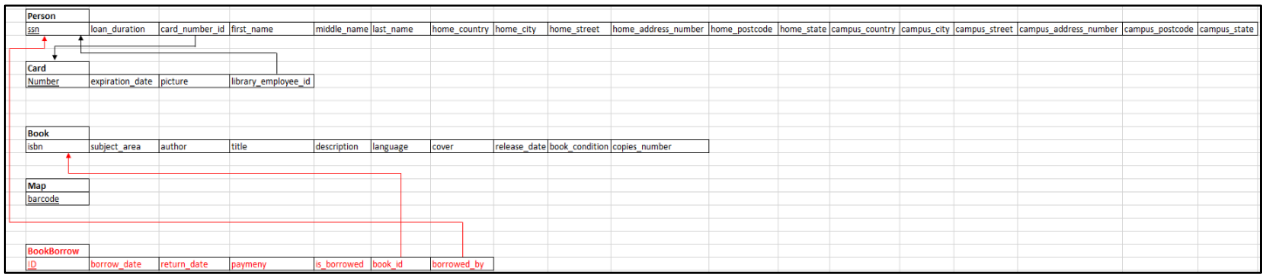


Figure 11 Database schema after fifth step of transforming of EER diagram

3.4.6 Step six: Mapping of multivalued attributes

In the sixth step it is required to create a new relation for each multivalued attribute. The created relation will include a value corresponding to the main entity plus the primary key attributes as a foreign key in the entity that contains the multivalued attribute. In the Georgia Tech Library EER diagram it is only Person entity that applies. The attribute is a Phone number. Consequently, a new table will be created for it and the table will be related to a Person schema.

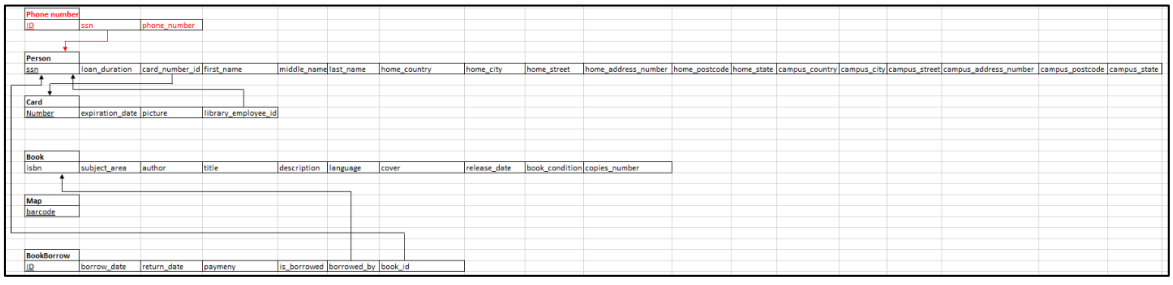


Figure 12 Database schema after sixth step of transforming of EER diagram

3.4.7 Step seven: Mapping of N-ary relationship types

During the step it is necessary to create for each n-ary relationship type R where n>2, a new relationship relation S to represent R. In the S relation the primary keys of the relations that represent the participating entity types should be included as well as any simple attributes of the n-ary relationship type as attributes of S. In the EER diagram there are no N-ary relationship types. Consequently, the step was skipped by us.

3.4.8 Step eight: Mapping specialization or generalization

This is the first of two steps strictly directed to EER diagrams. During the step, it is required to convert each specialization with m subclasses and the superclass where the attributes of the superclass are foreign keys relations to subclasses. However, there are multiple options that can be choose while mapping specialization and generalization. They are:

1. Multiple relations – superclass and subclasses.
   Creating a relation for a specific superclass and its attributes. Additionally, create relations for each subclass that contains local attributes plus the primary key of a superclass.
2. Multiple relations – subclass relations only
   Create a relation that contains attributes from all its subclasses. It works only for specialization whose subclasses are total. Which means that every entity in the superclass must belong at least to one of its subclasses.
3. Single relation with one type attribute
   Create single relation for each subclass. Each of the created subclasses will consist of common attributes and attributes specified for the subclass.
4. Single relation with multiple type attributes.
   Create a relation with all common and individual attributes and add special flags that will determinate to which subclass the specific tuple relates to.

For our generalization of library material, we have decided to use option 2. Consequently, we ended up having separate Map and Book entity to which an Added date attribute has been appended.

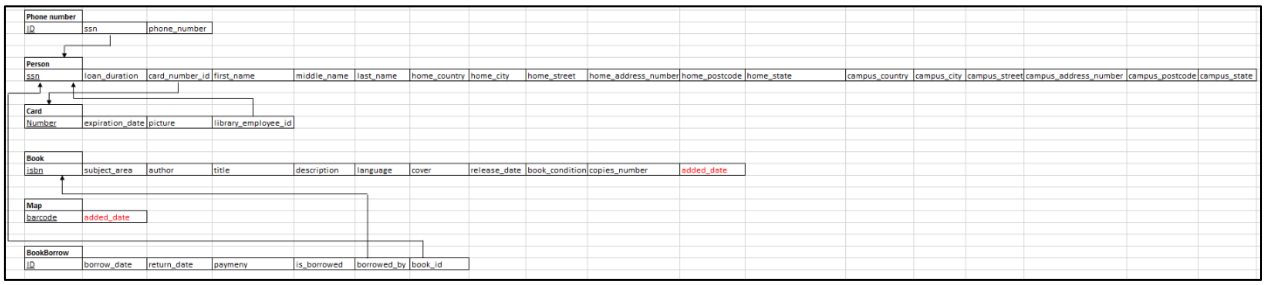


Figure 13 Resulting database schema after mapping a generalization of a library material

While choosing an option for the specialization of book between: “On loan book”, “Reference book” and “Rare book” entities, because there is only one additional attribute for the first of them, the most appropriate option 3. In result the no. avail. copies attribute was added to a book schema. Nonetheless, in order to distinguish different types of books, we have also decided to add one more column that will define what is the book’s type. On the other hand, we could use option four. However, on the longer go it would take much more memory if, a number of different types of book will increase in the future.

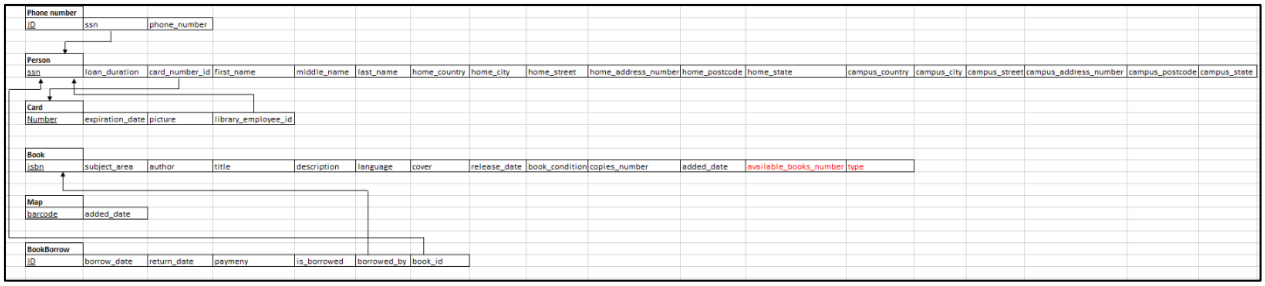


Figure 14 Resulting database schema after transforming book specialization

Another specialization that was take under consideration was of Person entity. It is an overlapping specialization, which means that a certain person can be for example both student and employee. However, it is not that common, so the best way to handle it is to use first option. Otherwise there would be a lot of redundant data if the second option would be chosen.

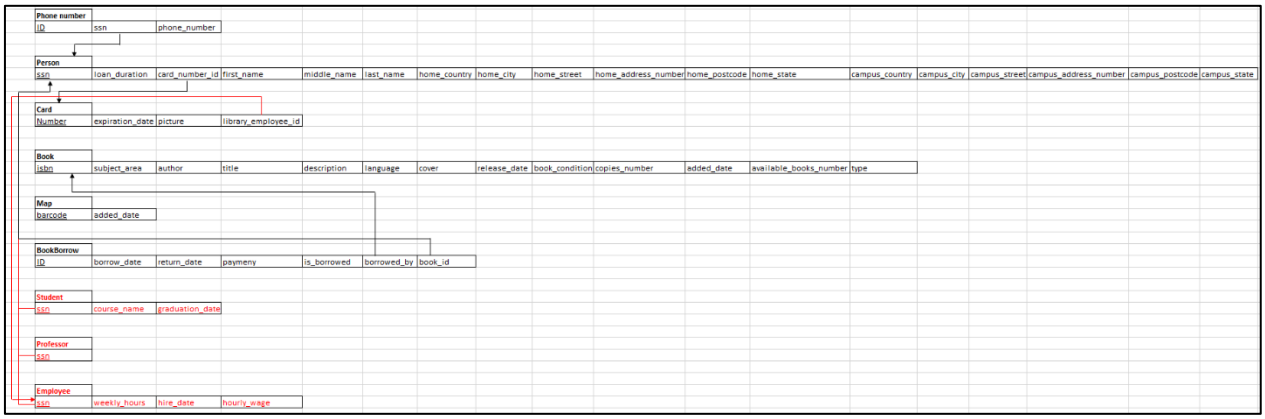


Figure 15 Resulting database schema after transforming person specialization

The last specialization is the one related to an employee. As it is known from the case description, there are different kinds of the library employees. Nonetheless our subclasses of an Employee entity do not contain any separate attributes. Consequently, we have applied an approach that somehow refers to the third option. Since no additionally attributes are present in subclasses, a new attribute which defines the employee type was added to the Employee entity.

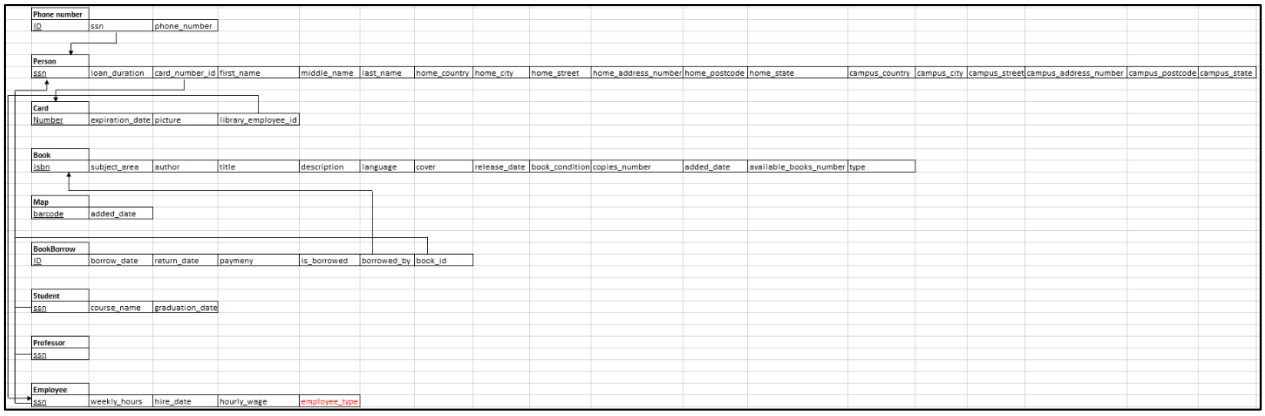


Figure 16 Resulting database schema after transforming employee specialization

3.4.9 Step nine: Mapping of union types (categories)

For mapping a category that defining superclass have different keys, it is customary to specify a new key attribute, called a surrogate key, when creating a relation to correspond to the category. There is no Union Types in the GTL EED, that is why there was nothing that was done to the schema from eight step.

3.5 Initial database schema

The following schema is a representation of what we have achieved by having transformed EER diagram.

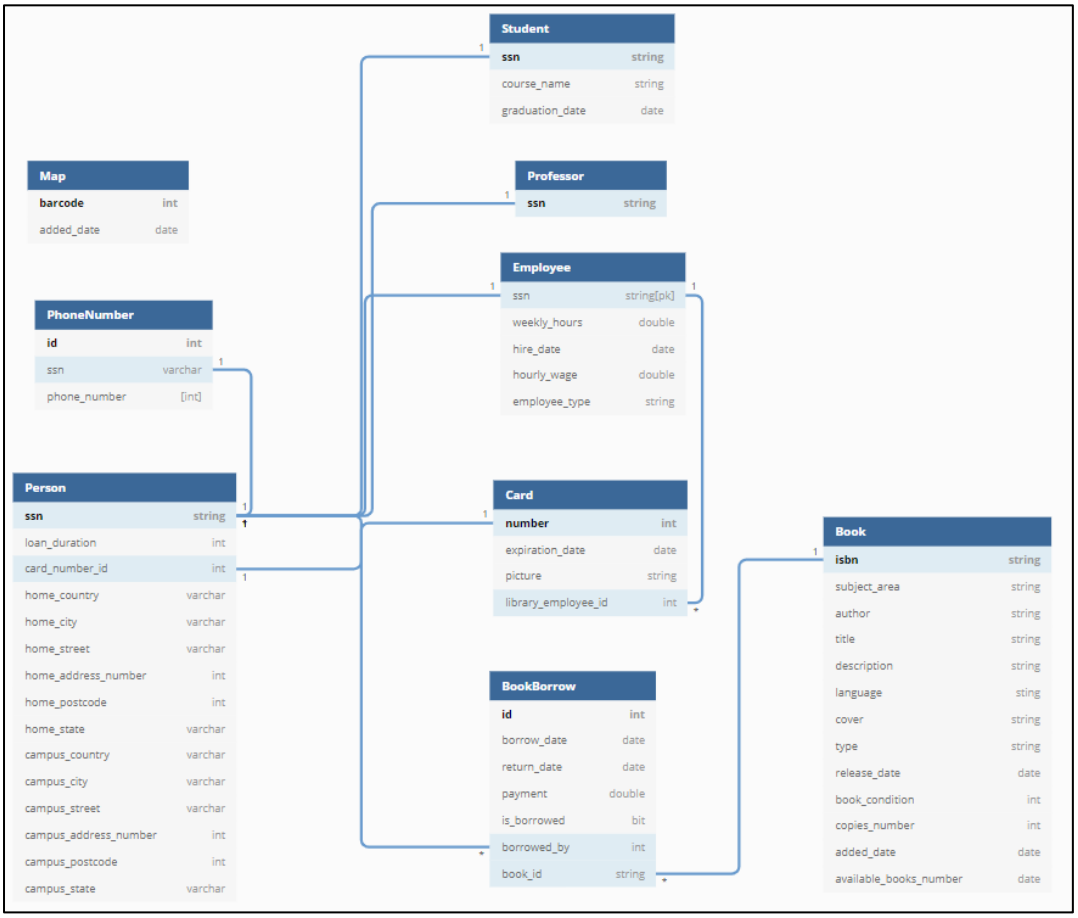


Figure 17 Initial database schema

After some considerations and discussions about different tables, before normalizing the schema, as a team we have decided to reconstruct it slightly. By taking each table we found some issues that we thought need to be resolved.

BookCatalog table

So far, we had a book table which stores information about a specific book. However, having mapped an EER diagram into database schema we found out that there might be a problem of storing multiple copies of the same book. Two same copies would have the same ISBN and the value is a primary key in the Book table. Consequently, it would not be possible to store for example five copies of the same book in the table. For that reason, we decided to create a table called BookCatalog where copies of a specific book are stored. The table has three columns which are: ID, ISBN, and book_condition. ID is a primary key and the ISBN is a foreign key reference to the ISBN primary key value in a Book table. The book_condition column

defines what is the condition of a book copy. Consequently, in a BookBorrow table a foreign key reference from Book table should be changed. Now it should be referencing BookCatalog, so that it is traceable what copy of a book is being borrowed.

BookCondition table

As it is known from a case description, books that are in a bad condition should not be available for borrowing. Because we have defined conditions which are: {Fine, Destroyed, Slightly worn}, the good practice was to create a new table that consists of id and the condition name. After that, the relation between BookCatalog and BookCondition tables was established. By doing that, we avoided inserting redundant data every time a copy of a book is added.

Having added the two tables described above, the database schema looks the following.

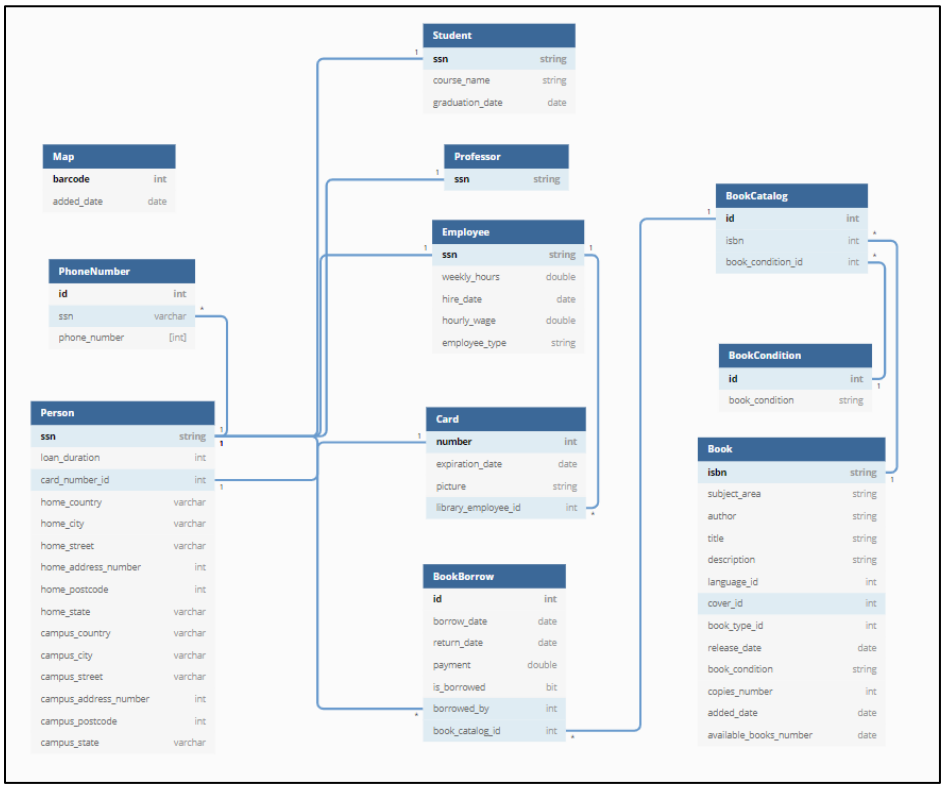


Figure 18 Database schema after creating BookCatalog and BookCondition tables

Book table

Some of the table's attributes are predefined. Which means there is a limited amount of values that they can have. The columns are:

Language: we decided to give the column predefined values. There is a distinct number of values that the Language column in book table can take, we would only limit it to values in the set: {Japanese, Spanish, Danish, English, Polish, Greek, Chinese}. Because of that we have created new table called Language and used its primary key to establish relation *Language/Book* where Language.id is a foreign key in Book table. By doing that, we can restrict what language can be chosen and avoid situation when a user misspells the language name.

Cover: there are four different cover types that can be chosen. These are: {paperback, digital, hardcover case wrap, hardcover dust jacket}. For the same reason as language, it is also better to create a new table with the names and relate it to the Book table.

Type: the same situation as for language or cover columns. Types of the books that are pre-defined from the case description are: {Rare book, Reference book, Loan book}. That is why, we have decided to create an individual table for this purpose and relate it to the Book table.

After the changes, the database schema is the following

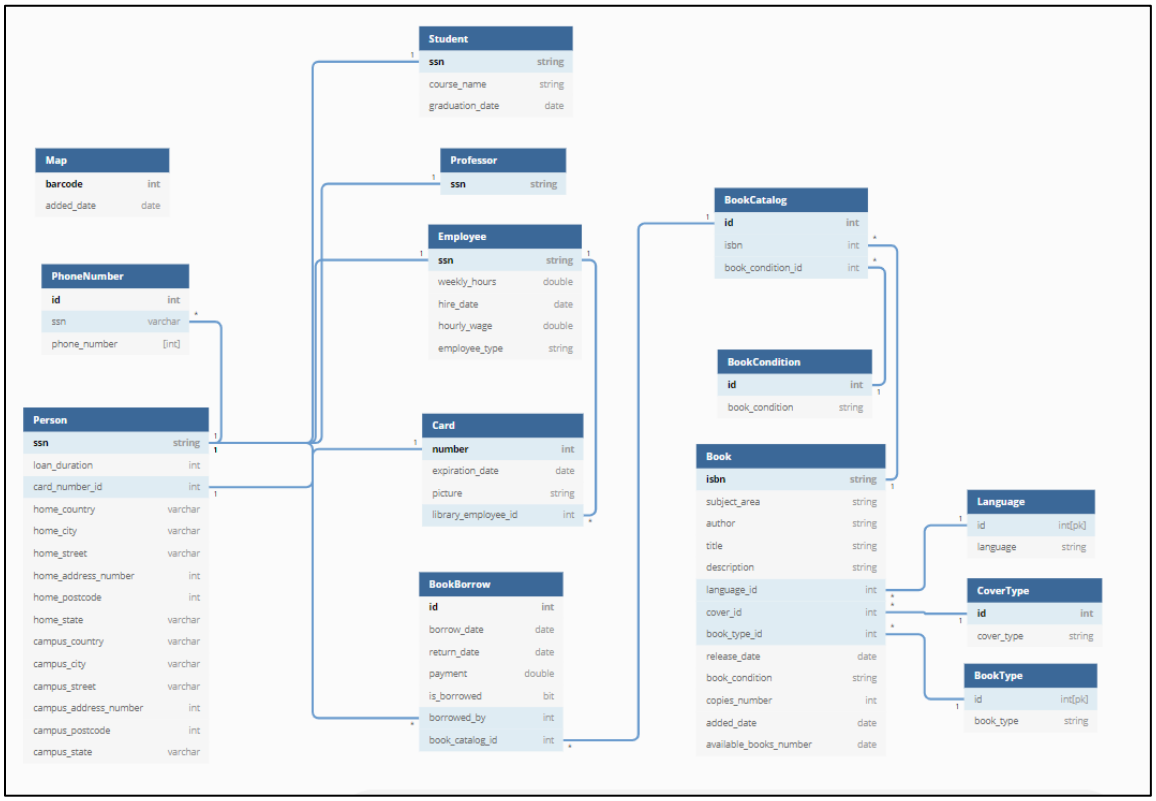


Figure 19 Database schema after creating Language, CoverType and BookType tables

Specialization of Person

The next thing that we have noticed is that whenever we would like to query to check if a certain person is a professor, employee, or a student then we would have to query at least four tables. Sometimes it would be tedious because there is not much cases when people have more than one role. We decided to create an additional table called PersonType. The table would store three ids representing roles of a specific person, which will be also have foreign key references to Professor, Employee, and a Student tables. Moreover, Person table would have a foreign key reference to the created table. Consequently, if we would like to determine the role of a certain person, then we would have to query only two tables which are Person and a newly created PersonType. In addition, we decided to add more information about Professor. Right now, there is only SSN stored in the table. The table was renamed to a “FacultyMember” and a column called faculty_member_type was added. It will be specifying what kind of faculty member is a certain person. The faculty_member_type’ column’s values will have values from the set: {Adjunct professor, Assistant professor, Full professor, Associate professor and Tenured}. Therefore, for the sake of data redundancy, we

have created a new table: FacultyMemberType where these values are stored. Consequently, FacultyMember table will be related to the newly made table by a foreign key constraint.

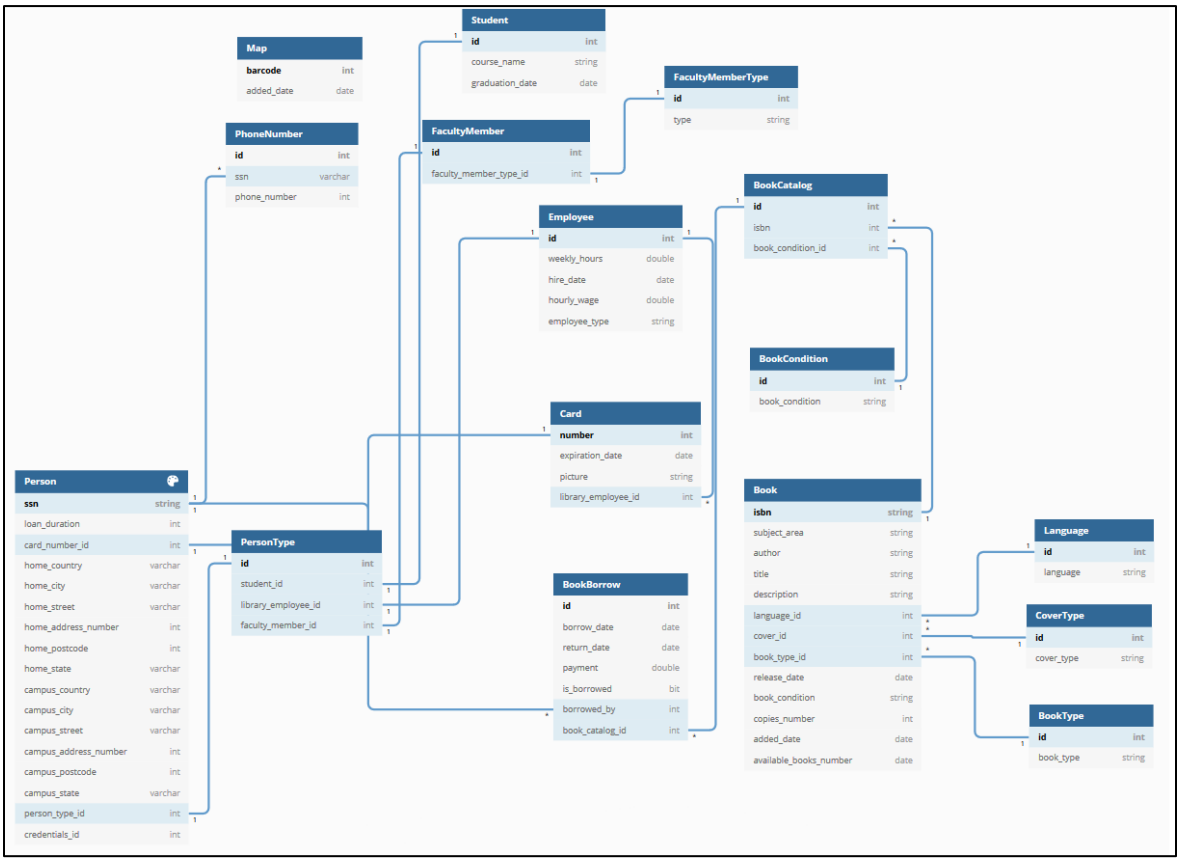


Figure 20 Database schema after creating a PersonType table

Credentials table

The last thing that we came up with before normalizing the database was to create a table where logins and password information for each person is going to be stored. It has been called Credentials. Each person has individual login and password. Moreover, it is essential to store what is the role of a user that sign in. It is because, in the implementation we will be able to use different database logins, depending on the role. The table was related to a Person one by a foreign key constraint. The primary key attribute of the newly created table is a login. That means there will not be two same users that have the same login value.

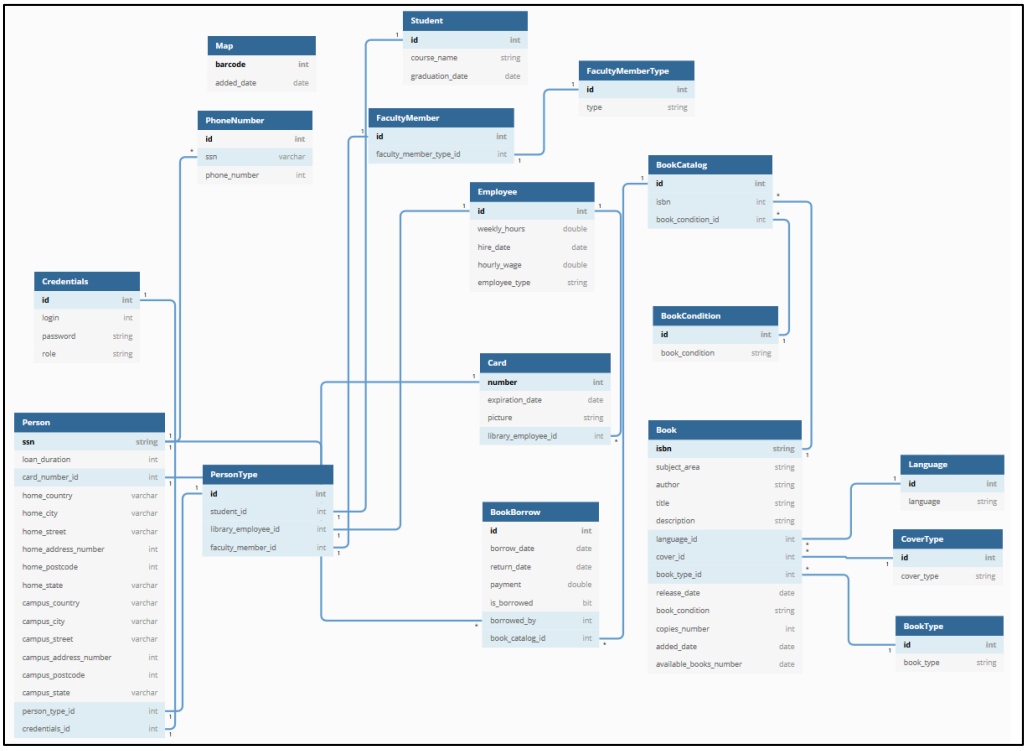


Figure 21 Database schema after creating a Credentials table

Having changed all the described steps, that we thought needed to be refactored after mapping the EER diagram, our database schema is the following.

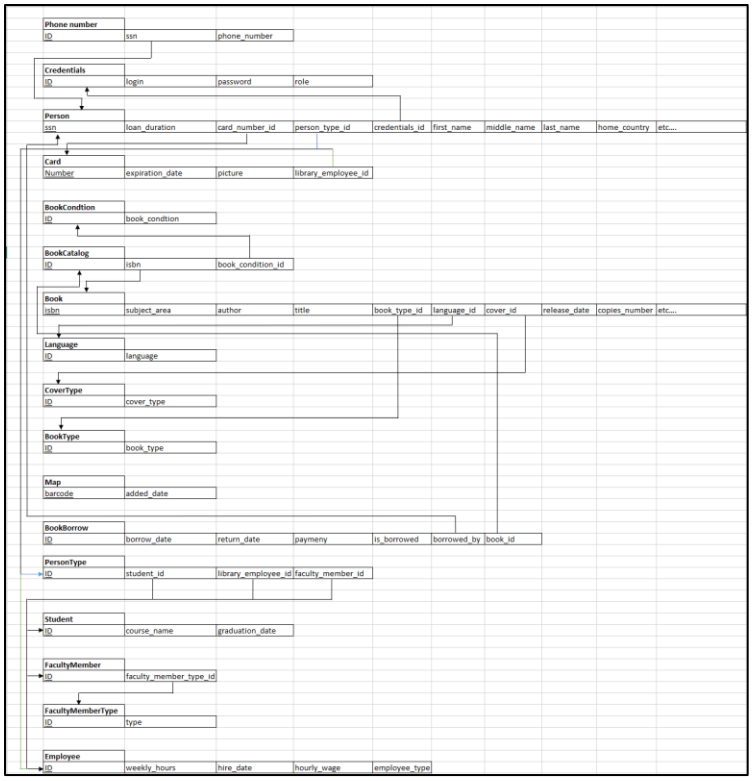


Figure 22 Refactored initial database schema with relations.

3.6 Normalization of a database schema

Normalization is a process of minimizing redundancy from a database. Redundant data may lead to some anomalies while performing such operation as insertion, deletion, or update. We have already refactored our initial database schema from the previous section so that the stored data would be less redundant. We achieved that for example by creating additional tables of Language, CoverType and BookType.

3.6.1 1NF

The relation fulfils first normal form if it does not contain any composite or multi-valued attribute. In addition, every attribute needs to be singled valued attribute. We can consider a table to be in 1st NF if the following conditions are satisfied:

1. Only single valued attributes are stored,
2. Domain of an attribute is stable,
3. Each column has a unique name,
4. Order of data does not matter.

All the table from our database schema meets the requirements which means that it is in 1NF.

3.6.2 2NF

In case of second normal form the database should be already in the first form and should not have any partial dependencies. We consider a table to be in 2nd NF if the following conditions are satisfied:

1. Already be in 1st NF
2. There shall be no partial dependency

Partial dependency occurs when one attribute of the table is dependent on the other one. As an example, we could have following StudentProject table. The ProjectName attribute and ProjectNo are in partial dependency. The name of the project is related to the project number. Therefore, having both features in this table is redundant and to avoid partial dependency this table must be changed.

<StudentProject>

StudentID	ProjectNo	StudentName	ProjectName
S01	199	Katie	Geo Location
S02	120	Ollie	Cluster Exploration

Figure 23 StudentProject table example not in 2NF

What can be done here is to create another table for project that would have primary key ProjectNO and ProjectName assigned to it. Then the StudentInfo table would no longer have partial dependencies on attribute ProjectName and ProjectNo, instead to get the name of the project we would join the StudentInfo table with ProjectInfo table using the ProjectNo.

<StudentInfo>

StudentID	ProjectNo	StudentName
S01	199	Katie
S02	120	Ollie

<ProjectInfo>

ProjectNo	ProjectName
199	Geo Location
120	Cluster Exploration

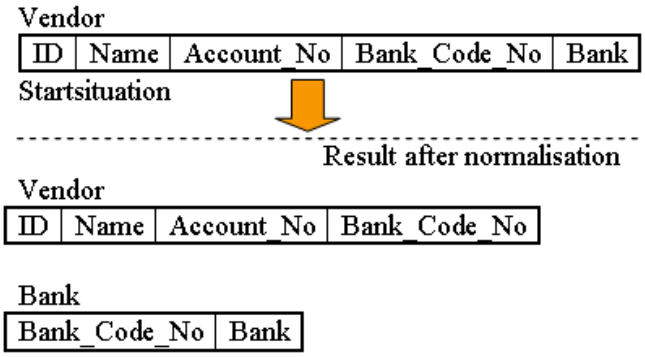
Figure 24 StudentInfo and ProjectInfo table being in 2NF

In case of our database composite primary key were not present. Each of our tables is uniquely identified by a single column key. This directly prevents us from having any partial dependencies in our database and guarantees us that if the database has already been put in the 1st Normalization Form it is automatically also in the 2nd Normalization Form

3.6.3 3NF

The table is in 3rd Normal Form when it is already in 2nd Normal Form and does not have transitive dependency. By its nature, the transitive dependency requires at least three attributes. The dependency occurs when a non-key attribute is transitively dependent on the primary key, which means it is dependent through another non-key attribute.

By giving an example we could have the following schema: (ID, Name, Account_No, Bank_Code_No, Bank). ID is a primary key and columns: Name, Account_No, Bank_Code_No are functionally dependent on the key. However, the Bank depends on Bank_Code_No. Therefore, there is a transitive dependency present between these two columns. To resolve that, it is necessary to split the table in two separate ones. As the following picture shows.



In case of our database we have noticed some transitive dependencies. Four tables should be refactored so that the database would be in 3NF. They are Person, BookBorrow, Employee and Student tables. First let`s look at the Person table and try to identify the transitive dependencies. In our case following attributes are:

1. Loan duration and person_type_id, the length with which certain book can be borrowed for, depends on the type of the person that is borrowing. Regular loan time for students is 21 days but faculty members can borrow it for 3 months. Therefore, we need to resolve this problem

2. Some of the address-related attributes have also transitive dependency. Country and state for both home and campus depend on appropriate postcode (which can be home or campus). It is possible to say what is the country and what is the state if a postcode is provided.
3. Postcode depends on city, street, and address number. Big cities might have multiple post numbers. That is why it is needed to have street name and address number to identify the correct post number.

Before normalization, our Person schema was:

Figure 25. Person schema before 3NF.

Having normalized it to 3NF, two new tables were created. Moreover, the loan_duration attribute was moved to the PersonType table and foreign keys references replaced partial dependencies in a Person schema.

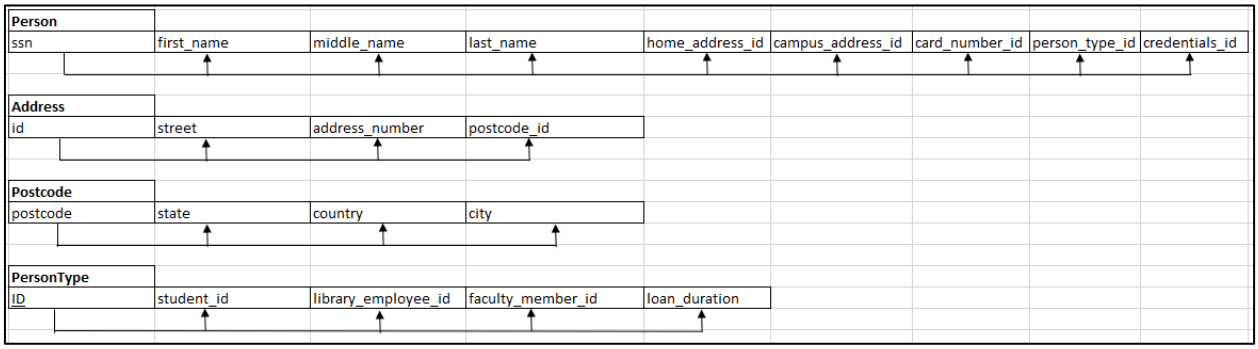


Figure 26 Resulting schemas after applying 3NF on the Person schema

We decided to create one Address table, instead of creating two tables for storing information about Home and Campus addresses. Then in a Person table, there are two attributes: home_address_id and campus_address_id that references to Address's id primary key constraint. In the Address table, there is a foreign key reference to a Postcode table, which contains postcode (primary key), state, country and city names.

Another table where we have detected transitive dependency is a BookBorrow table. In the table information about payment and if the book is still borrowed are stored. This information is partially dependent on return_date. If the return_date attribute has a value, then it is possible to calculate a payment and state that the book was returned.

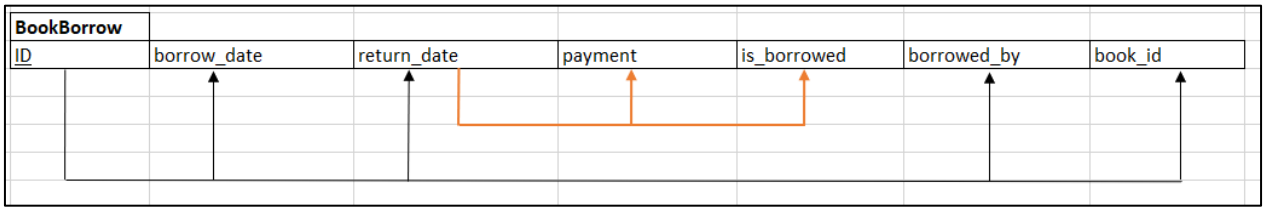


Figure 27 BookBorrow schema before 3NF.

To remove the partial dependency, it is necessary to create a new table. We have created a table called BookReturn that consists of ID (primary key constraint), return_date, payment and is_borrowed attributes. In addition, for the sake of calculating payment it is also necessary to store estimated_return_date to determine if the user have returned the book on time.

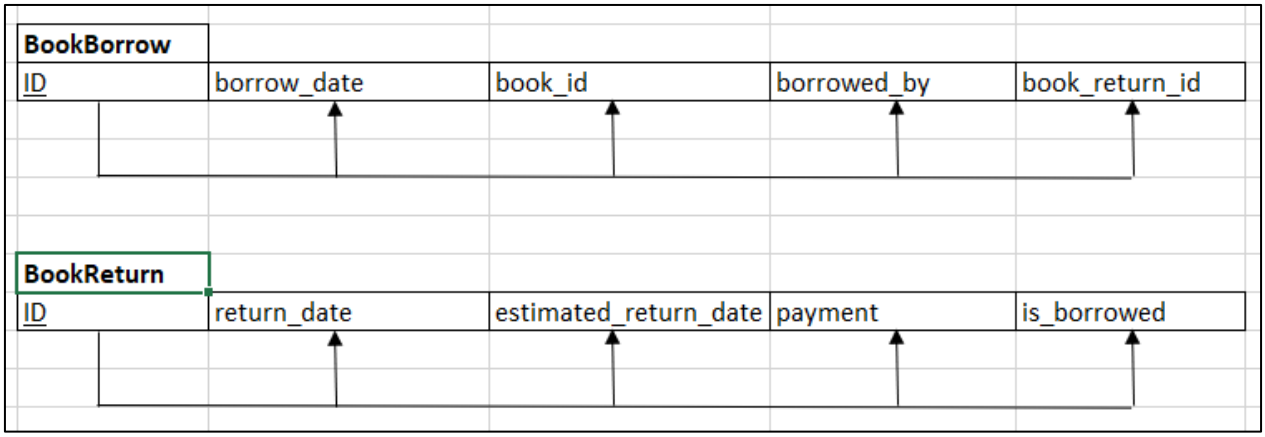


Figure 28 Resulting schemas after applying 3NF on the BookBorrow schema

The next table where a partial dependency was detected is an Employee table. It stores information about hourly_wage, weekly_hours, hire_date and employee_type. The partial dependency that occurs in the table is between an hourly_wage and an employee_type attributes. Hourly_wage depends on employee_type. For example, all reference librarians, earn the same amount of money for an hour of their work.

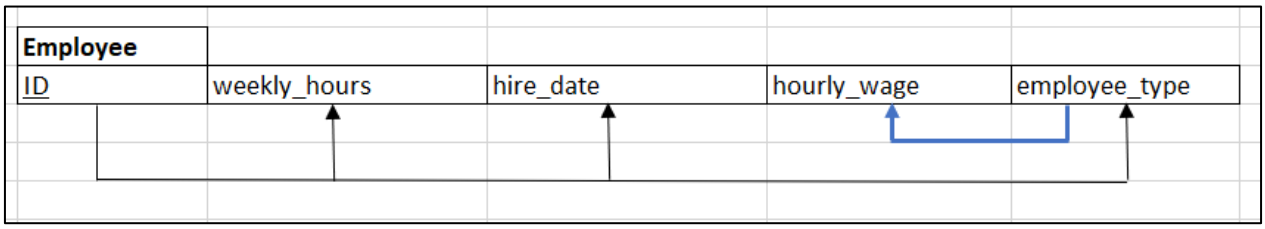


Figure 29 Employee schema before 3NF.

If we would like to remove the dependency, then it is necessary to create a new table to which the Employee table will be related. In result, a LibraryEmployeeType table has been created.

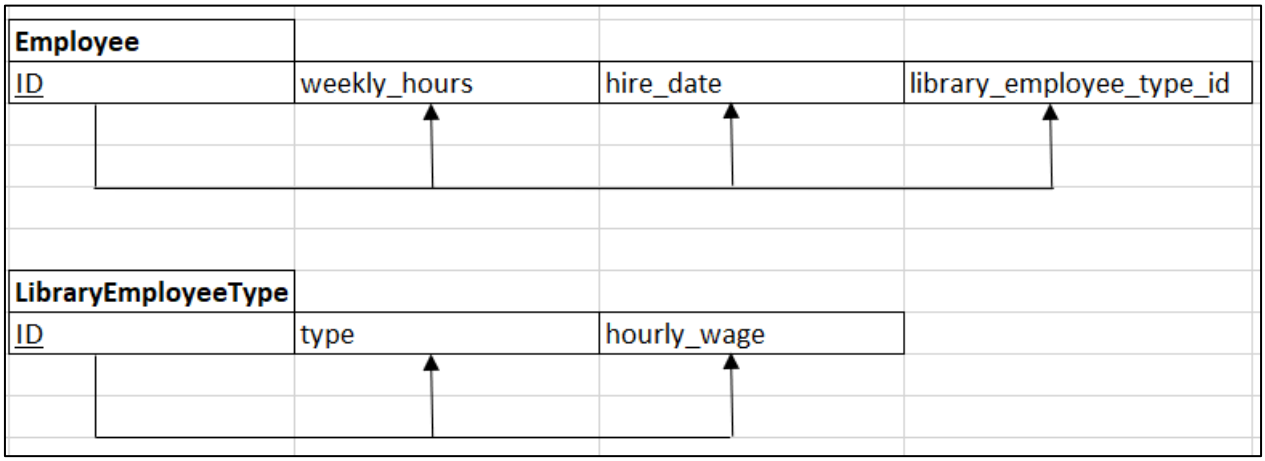


Figure 30 Resulting schemas after applying 3NF on the Employee schema

The last table that need to be refactored, so that our database would be in 3NF is a student table. The table stores information about course_name and graduation_date. In our opinion the graduation_date depends on

the actual course_name. It is pre-defined that for example a specific computer science course that starts on a certain date will finish on planned date.

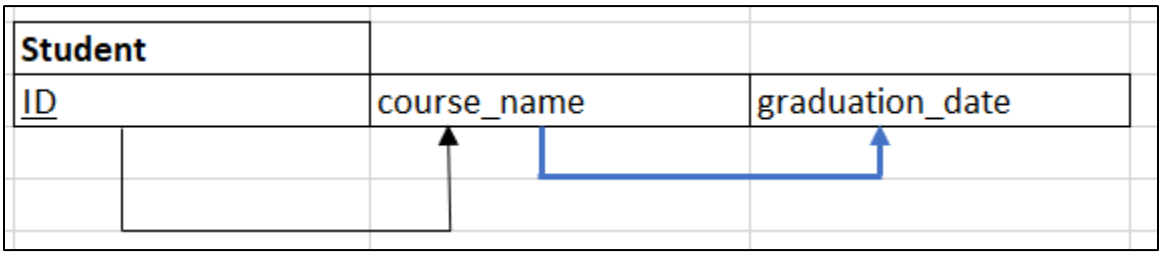


Figure 31 Student schema before 3NF.

In order to resolve it, a table called StudentType has been created and its primary key constraint was used to relate it to the Student table.

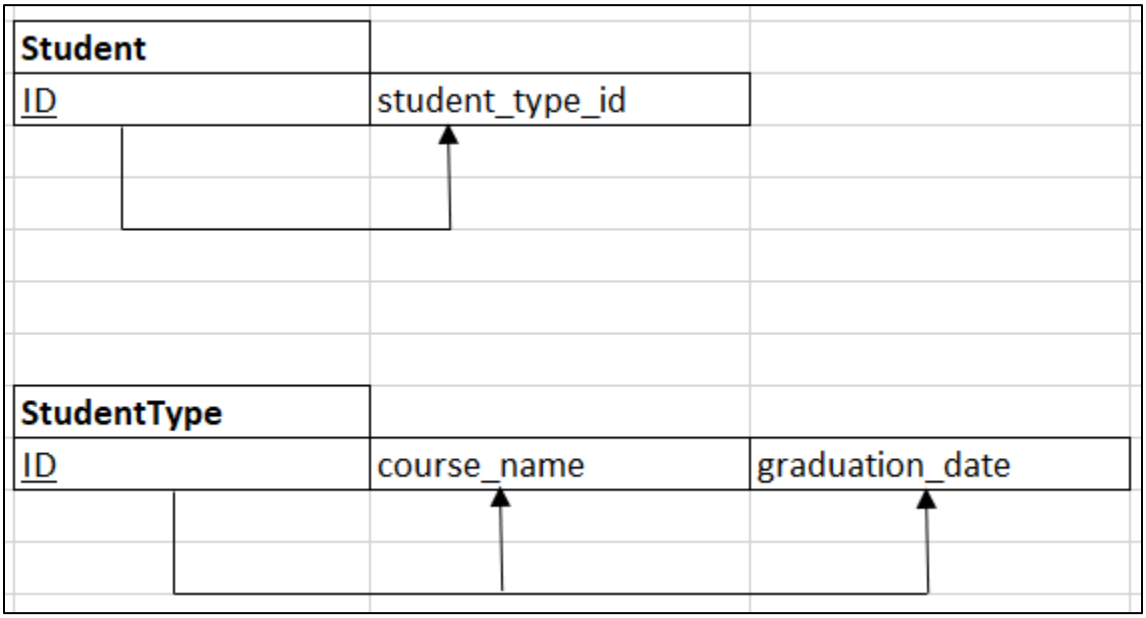


Figure 32 Resulting schemas after applying 3NF on the Student schema

Having transformed the database schema into 3NF, it looks as follows.

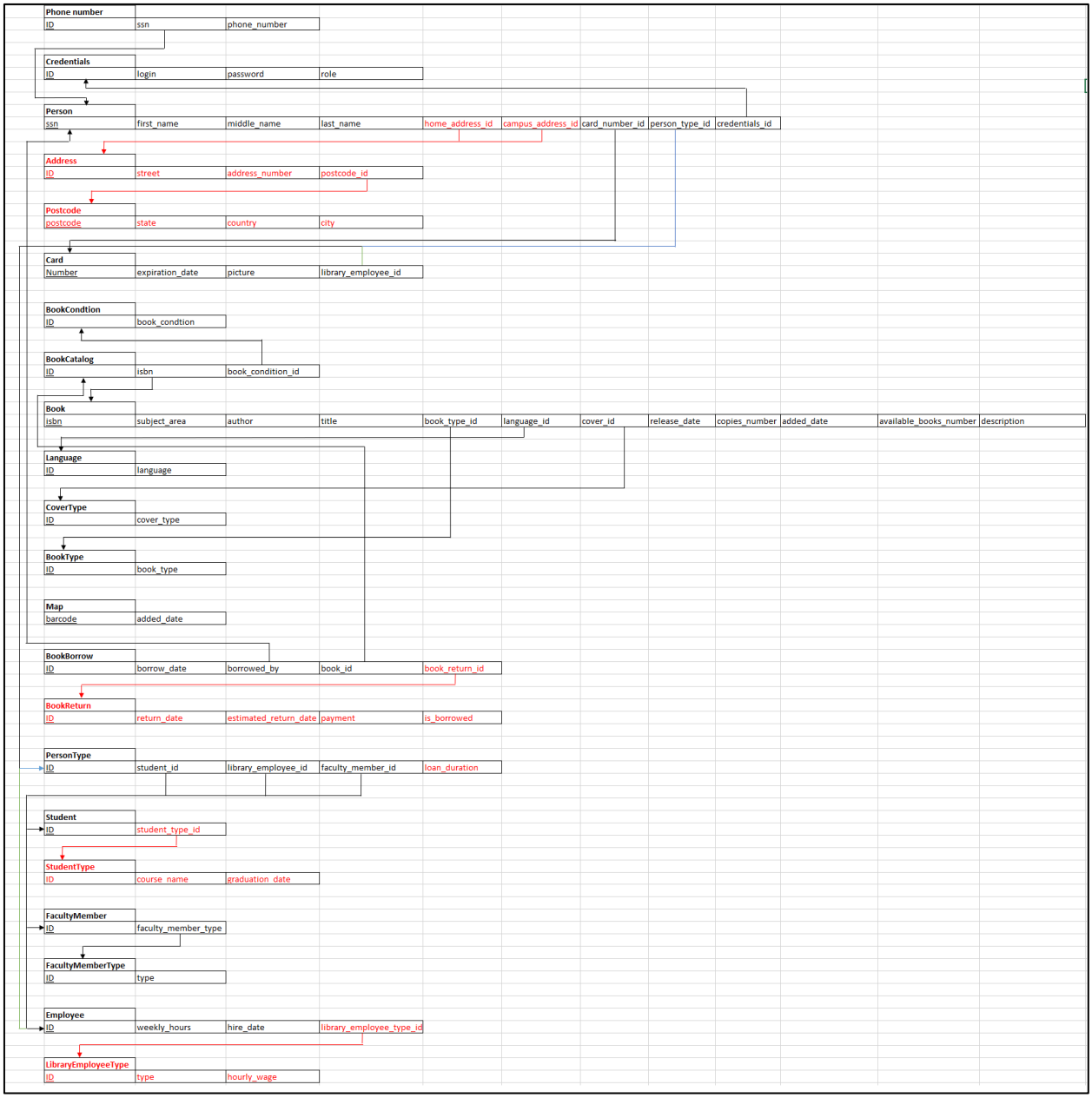


Figure 33 Resulting database schema in 3NF

3.6.4 BCNF

The last level of the normalization that we are going to reach in our database is Boyce-Codd Normal Form. It is an extension to the third normal form that we have already reached. For a table to be in BCNF it must fulfill the requirements:

1. Be in the Third Normal Form
2. For any dependency A->B, A should be a super key (A must not be a non-primary key if B is a prime attribute).

In the our 3NF form of a database, in each of our tables, their columns are dependent on the ID. They have already eliminated all transitive dependences. Therefore, all non-prime attributes are only depended on

primate column. There is no situation when it happened the other way. Consequently, our database fulfills the above requirements and it is in BCNF form.

3.7 Normalized database schema

At this moment we have a normalized database schema. That means we are ready for implementing it into a real database in a SSMS. At all, it contains twenty-two tables. Each of them is represented by a single valued primary key. Because the database is normalized, a number of redundant data that is going to be stored is reduced. Figure 33 shows how the database schema looks like after being normalized.

3.8 Implementation of a database

In the section, the implementation of each table in a SQL server is described. In includes quires for creation, adding constraints and limitations.

3.8.1 Credentials table

The credentials table stores information about users' logins. It is very important, because in the implementation it will be possible to take advantage from the data, in order to restrict API's endpoints access as well as choose the appropriate connection to a database.

```
CREATE TABLE dbo.[Credentials](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [login] [varchar](50) NOT NULL,
    [password] [varchar](100) NOT NULL,
    [role] [varchar](50) NOT NULL,
 CONSTRAINT [Credentials_pk] PRIMARY KEY NONCLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Figure 34 Credentials table creation script

The table consists of four columns. The primary key is the "id". It has identity characteristics which means that the value will be autogenerated and incremented, each time a new record will be added to database. None of the values in the table cannot be null because the presence of all the information in crucial. Despite the primary key, the values are varchars.

3.8.2 Book table

This is one of the most important tables. It stores data about books. What is meant by books is a certain edition rather than copy of the edition. That means that multiple book copies might have reference to one record in the table.

```
CREATE TABLE [dbo].[Book](
    [isbn] [varchar](13) NOT NULL,
    [description] [varchar](5000) NULL,
    [title] [varchar](100) NOT NULL,
    [author] [varchar](60) NOT NULL,
    [subject_area] [varchar](2000) NULL,
    [language_id] [int] NOT NULL,
    [released_date] [date] NOT NULL,
    [copies_number] [int] NOT NULL,
    [available_books_number] [int] NOT NULL,
    [cover_id] [int] NOT NULL,
    [book_type_id] [int] NOT NULL,
    [added_date] [date] NOT NULL,
 CONSTRAINT [PK_Book] PRIMARY KEY CLUSTERED
(
    [isbn] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Book]  WITH CHECK ADD  CONSTRAINT [FK_Book_BookType] FOREIGN KEY([book_type_id])
REFERENCES [dbo].[BookType] ([id])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[Book] CHECK CONSTRAINT [FK_Book_BookType]
GO

ALTER TABLE [dbo].[Book]  WITH CHECK ADD  CONSTRAINT [FK_Book_CoverType] FOREIGN KEY([cover_id])
REFERENCES [dbo].[CoverType] ([id])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[Book] CHECK CONSTRAINT [FK_Book_CoverType]
GO

ALTER TABLE [dbo].[Book]  WITH CHECK ADD  CONSTRAINT [FK_Book_Language] FOREIGN KEY([language_id])
REFERENCES [dbo].[Language] ([id])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[Book] CHECK CONSTRAINT [FK_Book_Language]
GO
```

Figure 35 Book table creation script

The table consists of twelve columns. Again, all of them must be have some value. The primary key is ISBN. It is because each edition of a certain book has a unique value of the number. It is possible to identify the book just by the number so there is no need for adding more values. The table has three foreign keys. They are language_id, cover_id and book_type_id, these keys are relating it with respectively Language, CoverType and BookType tables. Foreign keys have been created using a ON UPDATE CASCADE options. Consequently, the referencing rows are updated I the child table when the referenced row is updated in the parent table. The table also keeps track of number of copies of a specific books and number of available copies.

3.8.3 BookCatalog table

In the table stores different copies of books.

```
CREATE TABLE [dbo].[BookCatalog](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [isbn] [varchar](13) NOT NULL,
    [book_condition_id] [int] NOT NULL,
    [is_deleted] [bit] NOT NULL,
 CONSTRAINT [PK_BookCatalog] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[BookCatalog] ADD  DEFAULT ((0)) FOR [is_deleted]
GO

ALTER TABLE [dbo].[BookCatalog]  WITH CHECK ADD  CONSTRAINT [FK_BookCatalog_Book] FOREIGN KEY([isbn])
REFERENCES [dbo].[Book] ([isbn])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[BookCatalog] CHECK CONSTRAINT [FK_BookCatalog_Book]
GO

ALTER TABLE [dbo].[BookCatalog]  WITH CHECK ADD  CONSTRAINT [FK_BookCatalog_BookCondition] FOREIGN KEY([book_condition_id])
REFERENCES [dbo].[BookCondition] ([id])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[BookCatalog] CHECK CONSTRAINT [FK_BookCatalog_BookCondition]
GO
```

Figure 36 BookCatalog table creation script

Initially, the table was made of three columns: (id, isbn, book_conditon_id). The id is a primary key constraint and has identity. ISBN is a foreign key reference to a Book table and the last value is a foreign key that relates the table with the BookCondition one. As well as the Book table, foreign key has on update CASCADE option on. During the implementation, we decided to add is_deleted column. The column has type of bit. It determines whether or not is the copy of a specific book deleted.

3.8.4 BookCondition table

The table is very simple the only thing that is stores is pre-defined conditions of books. It is used by a BookCatalog to determine what is the condition of a specific copy.

```
CREATE TABLE [dbo].[BookCondition](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [book_condition] [varchar](20) NOT NULL,
 CONSTRAINT [PK_BookCondition] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Figure 37 BookCondition table creation script

The primary key of the table is id which has an identity value. The key is referenced by book_condition_id column from BookCatalog table.

3.8.5 BookType table

Book type table is a simple table storing pre-defined values about the type of the book. It is used to determine whether specific book can be lent out of the library

```
CREATE TABLE [dbo].[BookType](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [book_type] [varchar](30) NOT NULL,
 CONSTRAINT [PK_BookType] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Figure 38 BookType table creation script

The primary key of the table is id which has an identity value. The key is referenced by book_type_id column from Book table.

3.8.6 CoverType table

Cover type is a simple table storing pre-defined values about the cover of the book. It is used to determine the type of the cover that given book has.

```
CREATE TABLE [dbo].[CoverType](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [cover_type] [varchar](30) NOT NULL,
 CONSTRAINT [PK_CoverType] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Figure 39 CoverType table creation script

The primary key of the table is id which has an identity value. The key is referenced by Cover_id column from Book table.

3.8.7 Language table

Language table is a simple table storing pre-defined values about the language of the book. It is used to determine the type of the language with which the book was written.

```
CREATE TABLE dbo.[Language](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [language] [varchar](20) NOT NULL,
 CONSTRAINT [PK_Language] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Figure 40 Language table creation script

The primary key of the table is id which has an identity value. The key is referenced by Language_id column from Book table.

3.8.8 Map table

Map table is a resource table with 3 columns. Barcode that is unique identifier of the map, added_date stating when the map was added and is_deleted that states whether the map has been deleted from the library`s equipment.

```
CREATE TABLE [dbo].[Map](
    [barcode] [nchar](12) NOT NULL,
    [added_date] [date] NOT NULL,
    [is_deleted] [bit] NOT NULL,
 CONSTRAINT [PK_Map] PRIMARY KEY CLUSTERED
(
    [barcode] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [Book].[Map] ADD  DEFAULT ((0)) FOR [is_deleted]
GO
```

Figure 41 Map table creation script

The primary key of the table is barcode. The key is currently not referenced by any other table as we choose not to implement this kind of functionality within this iteration.

3.8.9 Card table

Card table is a table representing library card that an employee creates when users registers in library system. It has its own number, expiration date, picture, library_employee_id that is stating who have created this particular card and is_deleted, that is status of the card.

```
CREATE TABLE [dbo].[Card](
    [number] [int] NOT NULL,
    [expiration_date] [date] NOT NULL,
    [picture] [varchar](50) NULL,
    [library_employee_id] [int] NOT NULL,
    [is_deleted] [bit] NOT NULL,
 CONSTRAINT [PK_Card] PRIMARY KEY CLUSTERED
(
    [number] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Card] ADD  DEFAULT ((0)) FOR [is_deleted]
GO

ALTER TABLE [dbo].[Card]  WITH CHECK ADD  CONSTRAINT [FK_Card_LibraryEmployee] FOREIGN KEY([library_employee_id])
REFERENCES [dbo].[LibraryEmployee] ([employee_id])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[Card] CHECK CONSTRAINT [FK_Card_LibraryEmployee]
GO
```

Figure 42 Card table creation script

The primary key of this table is the number, which is an integer, picture for the time being is a varchar(30), but in future we would change this for instance disk location or other relevant solution. Then there is library_emplyoee_id which references employee_id from LibraryEmployee table. Last column - is_deleted is a bit which defines whether the card has been deleted.

3.8.10 Person table

Person table is a generic table defining all people in the library system. It is used to store information that each of the types of people in our database has as well as credentials.

```
CREATE TABLE [dbo].[Person](
    [ssn] [char](11) NOT NULL,
    [first_name] [varchar](30) NOT NULL,
    [middle_name] [varchar](30) NULL,
    [last_name] [varchar](30) NOT NULL,
    [home_address_id] [int] NOT NULL,
    [campus_address_id] [int] NOT NULL,
    [loan_duration] [int] NOT NULL,
    [card_number_id] [int] NOT NULL,
    [person_type_id] [int] NOT NULL,
    [is_deleted] [bit] NOT NULL,
    [credentials_id] [int] NOT NULL,
 CONSTRAINT [PK_Person] PRIMARY KEY CLUSTERED
(
    [ssn] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Person] ADD  DEFAULT ((0)) FOR [is_deleted]
GO

ALTER TABLE [dbo].[Person] ADD  DEFAULT ((1)) FOR [credentials_id]
GO

ALTER TABLE [dbo].[Person]  WITH CHECK ADD  CONSTRAINT [FK_Person_AddressCampus] FOREIGN KEY([campus_address_id])
REFERENCES [Information_Sensitive].[Address] ([id])
GO

ALTER TABLE [dbo].[Person] CHECK CONSTRAINT [FK_Person_AddressCampus]
GO

ALTER TABLE [dbo].[Person]  WITH CHECK ADD  CONSTRAINT [FK_Person_AddressHome] FOREIGN KEY([home_address_id])
REFERENCES [Information_Sensitive].[Address] ([id])
GO

ALTER TABLE [dbo].[Person] CHECK CONSTRAINT [FK_Person_AddressHome]
GO

ALTER TABLE [dbo].[Person]  WITH CHECK ADD  CONSTRAINT [FK_Person_Card] FOREIGN KEY([card_number_id])
REFERENCES [dbo].[Card] ([number])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[Person] CHECK CONSTRAINT [FK_Person_Card]
GO

ALTER TABLE [dbo].[Person]  WITH CHECK ADD  CONSTRAINT [FK_Person_Credentials] FOREIGN KEY([credentials_id])
REFERENCES [Admin].[Credentials] ([id])
GO

ALTER TABLE [dbo].[Person] CHECK CONSTRAINT [FK_Person_Credentials]
GO

ALTER TABLE [dbo].[Person]  WITH CHECK ADD  CONSTRAINT [FK_Person_PersonType] FOREIGN KEY([person_type_id])
REFERENCES [dbo].[PersonType] ([id])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[Person] CHECK CONSTRAINT [FK_Person_PersonType]
GO
```

Figure 43 Person table creation script

It has Primary key which is SSN, Foreign keys home_address_id and campus_address_id referencing Address table, card_number_id referencing Card table, person_type_id which reference the PersonType table and credencials_id referencing Credentials table. Person type is specialization table we implemented and specific option tables will be describe in following part of this document. It also has is_deleted bit column same as in previous examples.

3.8.11 PersonType table

PersonType table is a logical link between Person and its specializations (Student, Library, Employee).

```
CREATE TABLE [dbo].[PersonType](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [student_id] [int] NULL,
    [library_employee_id] [int] NULL,
    [faculty_member_id] [int] NULL,
 CONSTRAINT [PK_PersonType] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[PersonType]  WITH CHECK ADD  CONSTRAINT [FK_PersonType_FacultyMember] FOREIGN KEY([faculty_member_id])
REFERENCES [dbo].[FacultyMember] ([faculty_member_id])
GO

ALTER TABLE [dbo].[PersonType] CHECK CONSTRAINT [FK_PersonType_FacultyMember]
GO

ALTER TABLE [dbo].[PersonType]  WITH CHECK ADD  CONSTRAINT [FK_PersonType_Student] FOREIGN KEY([student_id])
REFERENCES [dbo].[Student] ([student_id])
GO

ALTER TABLE [dbo].[PersonType] CHECK CONSTRAINT [FK_PersonType_Student]
GO
```

Figure 44 PersonType table creation script

The primary key in this table is id which is used by mentioned previously Person table. Furthermore, it has 3 foreign keys that are used to retrieve person and its specialization student_id, library_employee_id, faculty_member_id. Using the specified id (either for student, employee, faculty member) we can then join one or more tables from set FacultyMember table, LibraryEmployee table and StudentTable.

3.8.12 PhoneNumber table

PhoneNumber table is a simple table that holds phone numbers related with specific person.

```
CREATE TABLE [dbo].[PhoneNumber](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [ssn] [char](11) NOT NULL,
    [phone_number] [char](7) NOT NULL,
 CONSTRAINT [PK_PhoneNumber] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[PhoneNumber]  WITH CHECK ADD  CONSTRAINT [FK_PhoneNumber_Person] FOREIGN KEY([ssn])
REFERENCES [dbo].[Person] ([ssn])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[PhoneNumber] CHECK CONSTRAINT [FK_PhoneNumber_Person]
GO
```

Figure 45 PhoneNumber table creation script

The primary key used in this table is SNN. If we wanted to list all the phone numbers for a person, we would first get its SNN and then look it up in PhoneNumber table.

3.8.13 Postcode table

Postcode table is simple table holding information about City, state and postcode for specific address.

```
CREATE TABLE [dbo].[Postcode](
    [postcode] [char](5) NOT NULL,
    [city] [varchar](45) NOT NULL,
    [state] [varchar](52) NOT NULL,
    [country] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Postcode] PRIMARY KEY CLUSTERED
(
    [postcode] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Figure 46 Postcode table creation script

Primary key in this case is clustered and it is postcode.

3.8.14 Student table

Student table is a table that we have previously joined as one of the options in PersonType table. It consists all the relevant information about student as well as reference to student_type.

```
CREATE TABLE [dbo].[Student](
    [student_id] [int] IDENTITY(1,1) NOT NULL,
    [student_type_id] [int] NOT NULL,
    [deadlines_missed] [int] NOT NULL,
    [gpa] [float] NOT NULL,
 CONSTRAINT [PK_Student] PRIMARY KEY CLUSTERED
(
    [student_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Student]  WITH CHECK ADD  CONSTRAINT [FK_Student_StudentType] FOREIGN KEY([student_type_id])
REFERENCES [dbo].[StudentType] ([id])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[Student] CHECK CONSTRAINT [FK_Student_StudentType]
GO
```

Figure 47 Student table creation script

Here we have student_id as a primary key. Student_type_id which references Student_Type table. The table also has information about GPA and number of deadlines missed.

3.8.15 StudentType table

StudentType table is a simple table storing information about the student.

```
CREATE TABLE [dbo].[StudentType](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [course_name] [varchar](50) NOT NULL,
    [graduation_date] [date] NOT NULL,
 CONSTRAINT [PK_StudentType] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Figure 48 StudentType table creation script

The primary key here is the ID and extra info about the student his graduation date and course name.

3.8.16 Address table

Address table is a simple table storing information about the address of a specific person.

```
CREATE TABLE [dbo].[Address](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [street] [varchar](50) NOT NULL,
    [address_number] [varchar](10) NOT NULL,
    [postcode_id] [char](5) NOT NULL,
 CONSTRAINT [PK_Address] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Address]  WITH CHECK ADD  CONSTRAINT [FK_Address_Postcode] FOREIGN KEY([postcode_id])
REFERENCES [dbo].[Postcode] ([postcode])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[Address] CHECK CONSTRAINT [FK_Address_Postcode]
GO
```

Figure 49 Address table creation script

It has primary key id, and foreign key postcode_id from table Postcode. Moreover, it has street and address_number.

3.8.17 FacultyMember table

FacultyMember table is a specialization table for a person (just like the student and library employee. So far it does not have much information inside it. But that is a subject to change once we get more features into the software.

```
CREATE TABLE [dbo].[FacultyMember](
    [faculty_member_id] [int] NOT NULL,
    [faculty_member_type_id] [int] NOT NULL,
 CONSTRAINT [PK_FacultyMember] PRIMARY KEY CLUSTERED
(
    [faculty_member_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[FacultyMember]  WITH CHECK ADD  CONSTRAINT [FK_FacultyMember_FacultyMemberType] FOREIGN KEY([faculty_member_type_id])
REFERENCES [dbo].[FacultyMemberType] ([id])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[FacultyMember] CHECK CONSTRAINT [FK_FacultyMember_FacultyMemberType]
GO
```

Figure 50 FacultyMember table creation script

It has a primary key used by PersonType table as well as foreign reference to FacultyMember type – faculty_member_type_id.

3.8.18 FacultyMember table

FacultyMemberType is a simple table storing additional information about the type of the faculty member.

```
CREATE TABLE [dbo].[FacultyMemberType](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [type] [varchar](40) NOT NULL,
 CONSTRAINT [PK_FacultyMemberType] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Figure 51 FacultyMemberType table creation script

It has a primary key id used by FacultyMember table and a description – type.

3.8.19 LibraryEmployee table

LibraryEmployee table is a table describing an employee of a library and his employment information. It is the last type of specialization of the Person table.

```
CREATE TABLE [dbo].[LibraryEmployee](
    [employee_id] [int] NOT NULL,
    [library_employee_type_id] [int] NOT NULL,
    [weekly_hours] [float] NOT NULL,
    [hire_date] [date] NOT NULL,
 CONSTRAINT [PK_LibraryEmployee] PRIMARY KEY CLUSTERED
(
    [employee_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[LibraryEmployee]  WITH CHECK ADD  CONSTRAINT [FK_LibraryEmployee_LibraryEmployeeType] FOREIGN KEY([library_employee_type_id])
REFERENCES [dbo].[LibraryEmployeeType] ([id])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[LibraryEmployee] CHECK CONSTRAINT [FK_LibraryEmployee_LibraryEmployeeType]
GO
```

Figure 52 LibraryEmployee table creation script

It has a primary key employee_id used by PersonType and reference to further details – library_employee_Type_id (same idea as with previous example) As well as information about weekly hours and hire date.

3.8.20 LibraryEmployeeType table

LibraryEmployeeType table is a table describing specific kind of an employee and their wage. All of the values are predefined. This is the table that in future could be used for invoicing.

```
CREATE TABLE [dbo].[LibraryEmployeeType](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [type] [varchar](50) NOT NULL,
    [hourly_wage] [float] NOT NULL,
 CONSTRAINT [PK_LibraryEmployeeType] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Figure 53 LibraryEmployeeType table creation script

Primary key in this case is id and we have also included text about the type and the hourly wage of an employee.

3.8.21 BookBorrow table

BookBorrow table is crucial from the libraries perspective as it binds specific SSN of the user with book_catalog_id moreover it stores reference to the return status of current loan (which will be described shortly).

```
CREATE TABLE [dbo].[BookBorrow](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [book_catalog_id] [int] NOT NULL,
    [ssn] [char](11) NOT NULL,
    [borrow_date] [date] NOT NULL,
    [book_return_id] [int] NOT NULL,
 CONSTRAINT [PK_BookBorrow] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[BookBorrow]  WITH CHECK ADD  CONSTRAINT [FK_BookBorrow_BookCatalog] FOREIGN KEY([book_catalog_id])
REFERENCES [dbo].[BookCatalog] ([id])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[BookBorrow] CHECK CONSTRAINT [FK_BookBorrow_BookCatalog]
GO

ALTER TABLE [dbo].[BookBorrow]  WITH CHECK ADD  CONSTRAINT [FK_BookBorrow_BookReturn] FOREIGN KEY([book_return_id])
REFERENCES [dbo].[BookReturn] ([id])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[BookBorrow] CHECK CONSTRAINT [FK_BookBorrow_BookReturn]
GO

ALTER TABLE [dbo].[BookBorrow]  WITH CHECK ADD  CONSTRAINT [FK_BookBorrow_Person] FOREIGN KEY([ssn])
REFERENCES [dbo].[Person] ([ssn])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[BookBorrow] CHECK CONSTRAINT [FK_BookBorrow_Person]
GO
```

Figure 54 BookBorrow table creation script

It has a primary key id. Reference to physical book that was lent out – book_catalog_id that references BookCatalog table.

3.8.22 BookReturn table

Book return table has information about current loan created by a user. It allows us to keep track of whether he has returned it yet or not, as well as estimated return date.

```
CREATE TABLE [dbo].[BookReturn](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [returned_date] [date] NULL,
    [estimated_return_date] [date] NOT NULL,
    [payment] [float] NOT NULL,
    [status] [bit] NOT NULL,
 CONSTRAINT [PK_BookReturn] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Figure 55 BookReturn table creation script

Here we have primary key id that is referenced by BookBorrow. Estimated return date, payment and status that defines whether the book is already returned or not.

3.9 Five useful queries

One of the tasks we were given during the project was to design five different queries and test their performance. To save the only resource that we have – time, our primary goal was to design queries which will be significant for us in later stages of the development process. Together with the group we decided to create a set of views which will allow us to fetch all required data without complex logic in the software. This process will be described in further parts of this report. As a next step, before we start to design the queries, we had to decide which information we would like to retrieve and use in our software. Referring to the database design from above some of the most significant fields are placed in different tables and they need to be joined together.

In SQL by the definition *"JOIN" clause is used to combine rows from two or more tables, based on related column between them.* We distinguish between several different join operations which are suitable for different requirements.

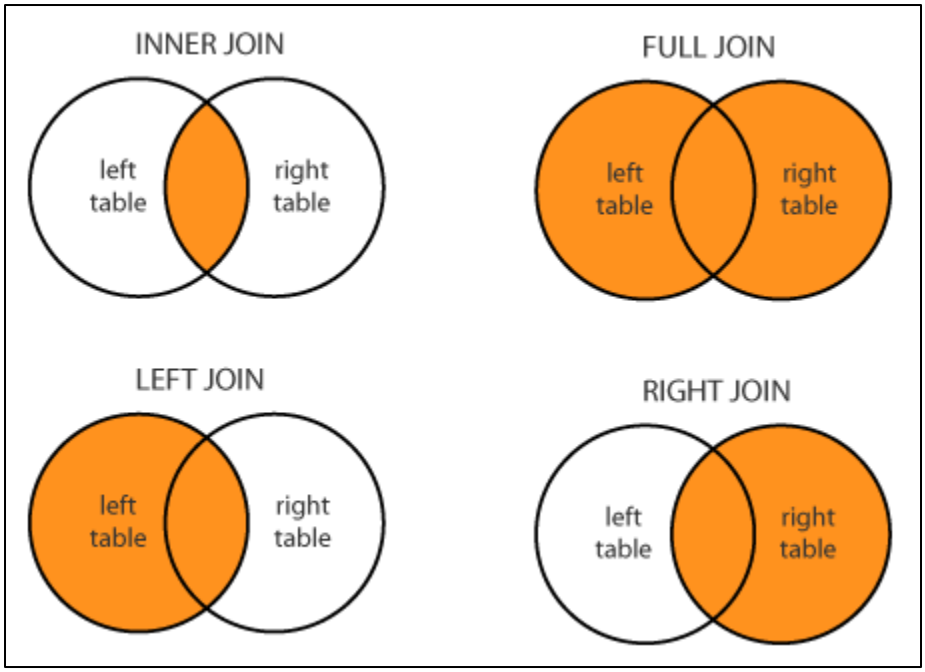


Figure 56 Visualisations of different joins

3.9.1 Loan Activities History

Equipped with this knowledge our group was ready to start the design of queries we find useful for our project. Our first choice was to retrieve information about all the loan activities which includes borrowing and returning of a specific book by one person.

	ssn	borrow_date	isbn	estimated_return_date	returned_date	payment	status
1	321-30-9427	2019-04-19	1615227248684	2020-04-10	2020-04-08	0	1
2	459-56-4777	2019-08-25	3355933265239	2019-09-15	2019-09-06	0	1
3	684-38-6511	2019-07-10	1149733462322	2019-07-31	2019-07-21	0	1
4	274-83-6067	2019-11-01	1237825496438	2019-11-22	2019-11-13	0	1

Figure 57 Result of the loan activities history query

The data we are interested in is social security number of a person which is part of BookBorrow table together with date of borrowing. Secondly, we join two tables which hold all information about the book

and its status which are BookReturn and BookCatalog. As a product we receive whole history of the library actions which can be filtered later in our software. The query is designed as follows:

$$\pi_{\text{ssn, borrow\_date, isbn, estimated\_return\_date, returned\_date, payment, status}} \text{BookBorrow} \bowtie_{\text{BookReturn.id = BookBorrow.book\_return\_id}} (\pi_{\text{id, estimated\_return\_date, returned\_date, payment, status}} \text{BookReturn}) \bowtie_{\text{BookBorrow.book\_catalog\_id = BookCatalog.id}} (\pi_{\text{id, isbn}} \text{BookCatalog})$$

Figure 58 Algebraic form of the loan activities history query

After the implementation, the query looks as follows:

```
SELECT BB.[ssn], BB.[borrow_date], BC.[isbn], BR.[estimated_return_date], BR.[returned_date], BR.[payment], BR.[status]
FROM Loan_Activities.BookBorrow BB
INNER JOIN (SELECT [id], [estimated_return_date], [returned_date], [payment], [status]
FROM Loan_Activities.BookReturn) BR ON BR.id = BB.book_return_id
INNER JOIN (SELECT [id], [isbn] FROM Book.BookCatalog) BC ON BB.book_catalog_id = BC.id
```

Figure 59 The loan activities history query in SQL

As the figure shows in the first step, we declare all the fields from every table we want to retrieve. Going further we perform inner join of BookBorrow and BookReturn tables on the fields: BookReturn.Id and a foreign key BookBorrow.book_return_id. At this stage we are provided with the ssn, date of the borrowing, payment and returning dates. To be able to identify the book by its isbn, we must perform one inner join more with the BookCatalog table on the BookBorrow.book_catalog_id foreign key and BookCatalog.id field.

Performance

Next part of the task was to test the performance of our query and try to improve it by applying non-clustered index on selected fields. By the definition it is an index which does not match the physical order of the data on the disk. Instead it is ordered by the columns. In the non-clustered index, the "*leaf*" does not contain actual data but a pointer to a place where the information is located so to a clustered index. This practice is beneficial especially in case when the data gets larger in the manner of performance. It allows data engine to locate the right information quickly without the need of scanning entire tables.

The non-clustered index must be considered adding in two cases. One of them is when the database must handle a set of columns used in the WHERE clause. Second the index besides the clustered one will speed up the operation. Next main scenario where non-clustered index should be used is when the data we want to return frequently must be ordered in a specific way. Additional sorting will not need to be done and the amount of CPU and memory utilization will be lowered.

To correctly identify the places where the indexes should be implemented in our database at first, we need to analyze the flow of operations our query performs. Therefore, for each of them execution tree was created. It helps to visualize the process and decide on actions which can improve it.

The first query flow:

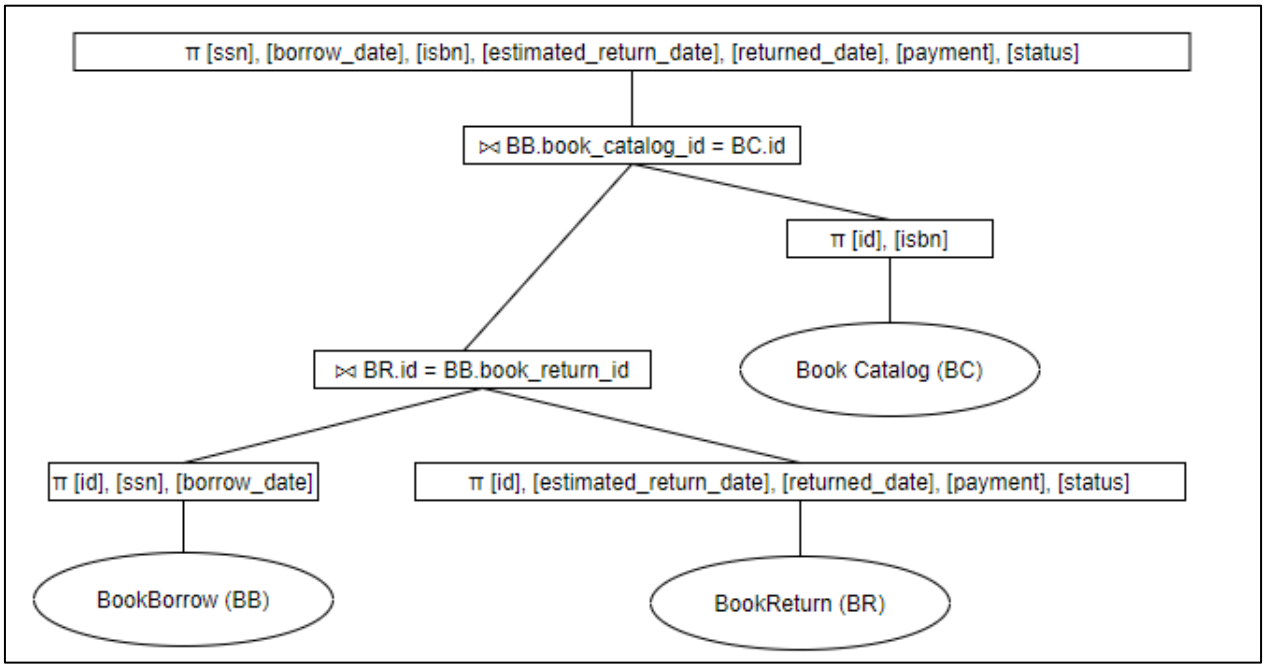


Figure 60 Execution plan of the loan activities history query

As the figure shows in the query there are no "WHERE" clauses. Furthermore, it does not require to return the list in any specific order. Consequently, there is no option of improving its performance by applying non-clustered index on any of returned fields. However, to experiment we decided to try implementing indexing on different fields. By examining the execution plan which is possible to display in Microsoft SQL Server Management Studio we noticed that our changes do not change the flow if operation.

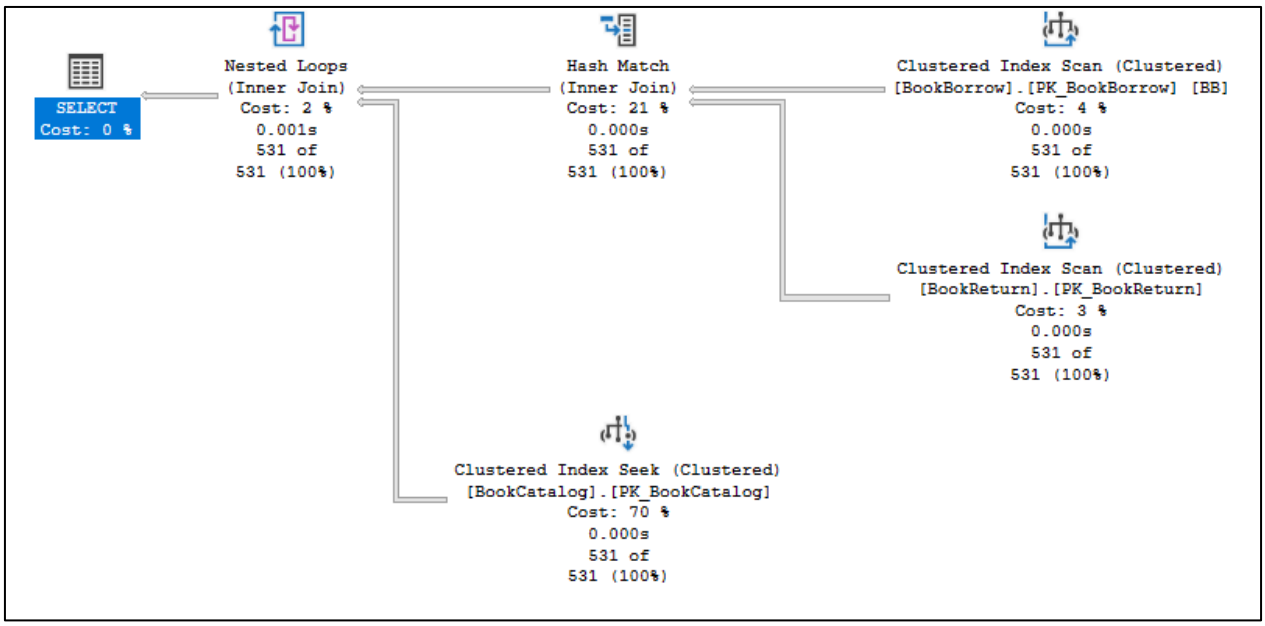


Figure 61 Execution plan of the loan activities history query in SQL

After examination we noticed that in this specific case every search field that the SQL loops through is a primary key and therefore the non-clustered index is redundant.

The execution plan gives us another important information about our query which is what kinds of joins does the SQL server performs to optimize the data retrieval. There are three kinds that are used: Nested Loops, Merge Join and Hash Match. Each of them differs regarding the level of complexity and therefore uses different amount of resources and time to execute.

Nested Loop

The first kind of joins we encountered were Nested Loops. The nested join compares each row of the "*Outer*" table to each row from the "*Inner*" table looking for the right records which match the join predicate. In the case of our query the fields which are compared are book_catalog_id from BookBorrow table and id form BookCatalog. The total number of rows compared with each other and the cost of the operation performed is proportional to the size of the outer table multiplied by the size of the inner table. The complexity of this operation can be described as:

$$O(NlogM)$$

The SQL Server optimizer might choose a Nested Loops when the one of the desired tables to joins is significantly small. Consequently, it requires fewest comparison and minimal I/O.

Hash Match

The next kind of join we noticed during the examination of execution plan is Hash Match. It is normally used when the input tables are large, and no adequate indexes exist on them. The join is performed in two phases. First of them is Build phase and second is Probe. The SQL server during these phases work on different tables that are desired to join.

In the build phase all the joining keys from the smaller table are being scanned. Going further a hash map in a memory is created holding previously read and hashed data.

The Probe phase similar to the build phase, scans the joining keys of the bigger table and hashes them with the same algorithm. Furthermore, it compares those with already existing data from the hash table, searching for a match. A hash function requires a significant amount of resources such as CPU cycles and memory to generate hash table. Depending on the resources available for the hash match to use, the SQL Server uses three different types: in-memory, grace or recursive hash join. However, as the purpose of this report is to describe the process we went through during the final project, we will not go into the details.

In the query described above based on the execution plan SQL server performed hash match before the nested loops. It occurred while joining the BookBorrow and BookReturn tables on book_return_id and id fields. The complexity of described join operation is:

$$O(N*h_c+M*h_m+J)$$

Merge Join

The last join operation our group can encounter is Merge Join. The first and the most important condition for the SQL server to use this kind of join is that it requires both input tables to be sorted on join columns. Going further thanks to pre-sorted data the process of matching begins immediately. The merge join reads a row from one input and compares it with the row of another table.

The merge join performs well while working with big inputs which are sorted, and the cost is the sum of rows from both tables.

$$O(N+M)$$

3.9.2 Personal information of students

Our next choice was to retrieve the most significant information about all the students registered in the library. This data is crucial for administrational purposes.

	first_name	middle_name	last_name	card_number_id	student_id
1	Dorothy	NULL	Zuniga	1025435856	1107401248
2	Joe	NULL	Sampson	1026750093	1107400789
3	Bart	Aaron Prince	Bennet	1027294938	1107401831
4	Frances	NULL	Herrera	1027605966	1107400668
5	Monica	NULL	Donaldson	1028649920	1107401432

Figure 62. Result of the students' personal information query

There are several operations necessary to achieve this result. The fields we are interested in are in two different tables Peron and PersonType. From the first one we select name, middle name, and last name together with card number. To retrieve student id the inner join must be performed. After the discussion in our group we designed the query as follows:

$$\pi_{\text{first\_name, middle\_name, last\_name, card\_number\_id, student\_id}} \text{Person} \bowtie_{\text{PersonType.id = Person.person\_type\_id}} (\pi_{\text{id, student\_id}} \sigma_{\text{student\_id} \neq \text{null}} \text{PersonType})$$

Figure 63 Algebraic form of the students' personal information query

Next step we took was to implement the query using SQL:

```
SELECT P.[first_name], P.[middle_name], P.[last_name], P.[card_number_id], PT.[student_id]
FROM Information_Basic.Person P
INNER JOIN (SELECT [id], [student_id] FROM Information_Basic.PersonType WHERE student_id IS NOT NULL) PT
ON PT.id = P.person_type_id
```

Figure 64 The personal students' information query in SQL

Similar to the flow of previous query described above the first step is to declare which fields do we want to display in our final result. Next step the code performs is an inner join of Person and PersonType tables on PersonType.id primary key and Person.person_type_id foreign key. An important part of this query is the "WHERE" clause which filters the PersonType table and retrieves only records with not empty student_id field.

Performance

To identify potential performance improvements, we created an execution tree:

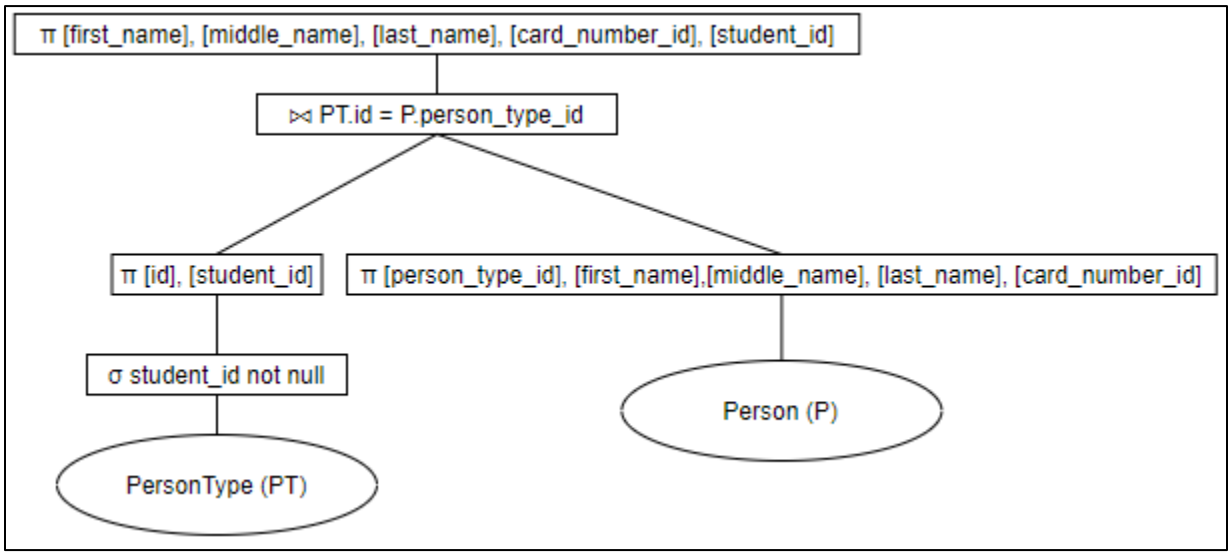


Figure 65 Execution plan of the personal students' information query

Following the rules described above our group discussed possible implementations of non-clustered indexes. Although, we do not return the list in any specific order, the query contains one "WHERE" clause which filters the result form PesonType table. As the filtering is performed on student_id field we decided to create an index on this field and test the performance.

To have a result which can be compared later before implementing the indexes we examined the execution tree:

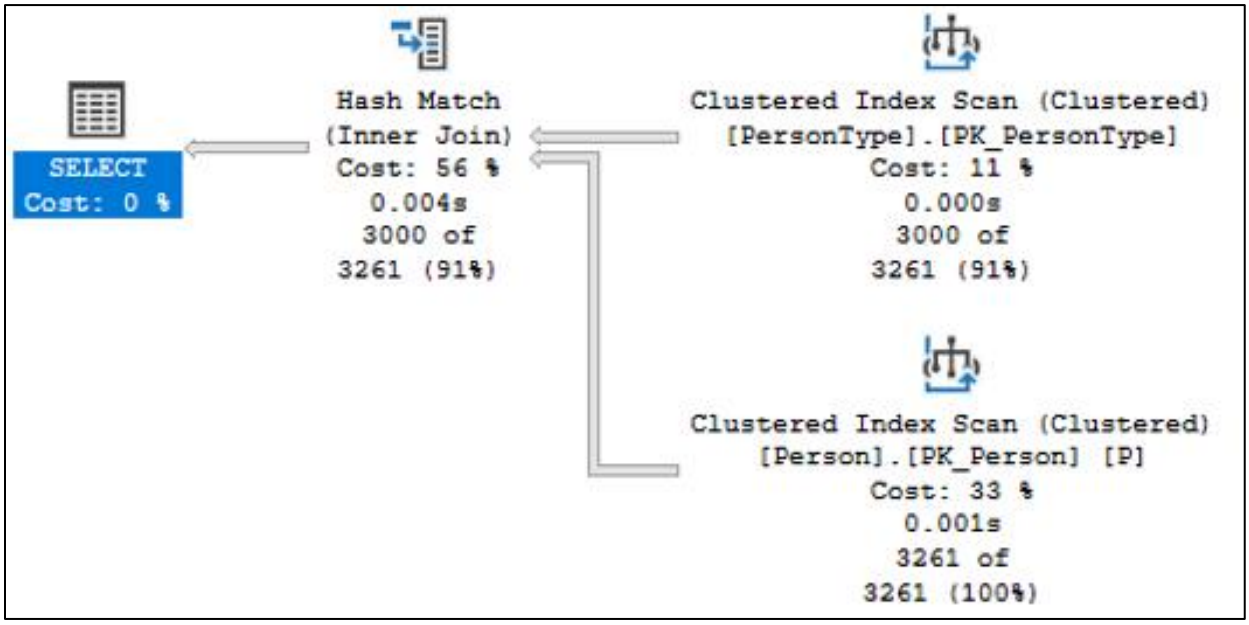


Figure 66 Execution plan the personal students' information query in SQL

After the implementation of the non-clustered index the result tree has changed:

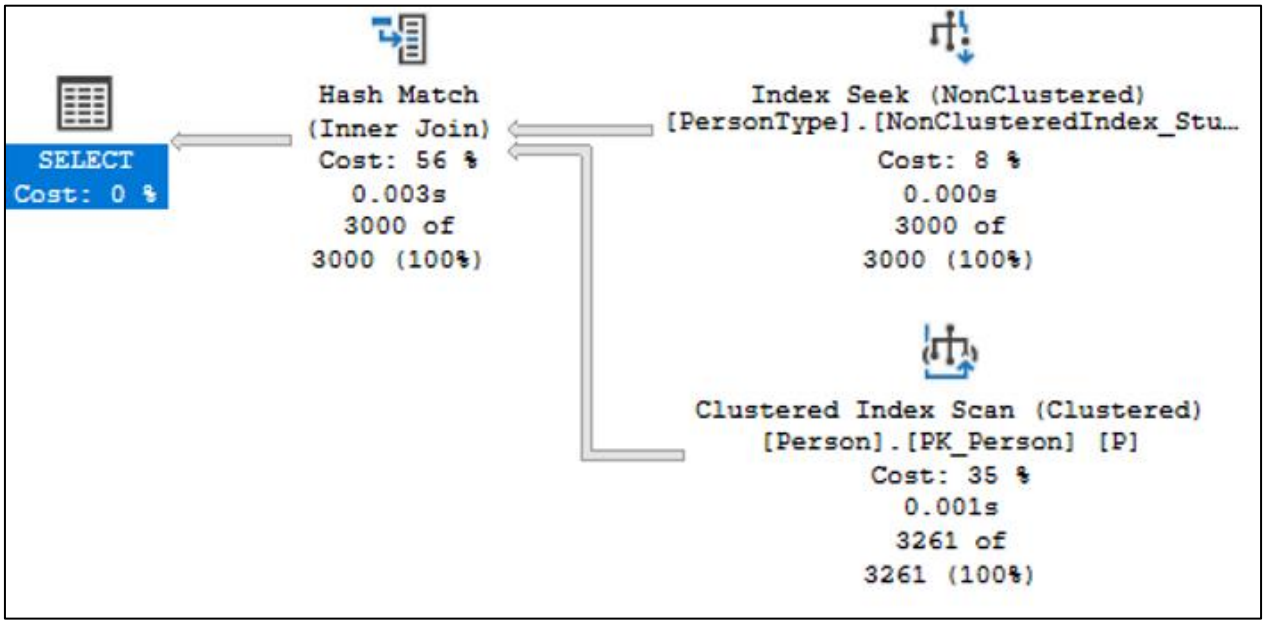


Figure 67 Execution plan the personal students' information query in SQL with non-cluster index

As the figure shows the search on PersonType table changed from "Clustered index scan" to "Index Seek". Microsoft Management Studio does not measure time below one millisecond and therefore there is no visible difference in the time of execution. However, by looking on the cost, which is the amount of resource this operation took, we can see the performance boost. The query consumes less resources after the implementation of non-clustered index. Our assumption is that if the data grows in the future the difference in time if execution will be more significant. In this case as it is visible, the SQL server chooses to use Hash Match join.

3.9.3 Personal information of employees and their wages

As an employer it is important to have records of the employees together with their personal information and wages. Therefore, the next query our group decided to create contains such information.

	first_name	last_name	employee_id	hourly_wage	weekly_hours	type
1	Gabrielle	Mendoza	37	31	32	Departmental associate librarians
2	Quentin	Pollard	29	35	32	Reference librarians
3	Devon	Mayer	85	19	32	Check-out staff
4	Ronda	O'Connell	38	45	36	Chief librarian
5	Evan	Santiago	48	35	36	Reference librarians

Figure 68 Result of the personal information of employees and their wages query

To correctly retrieve all the data needed several tables must be joined together. The first_name and last_name fields are stored in the Person table. Next information such as employee_id and weekly_hours are located in LibraryEmployee table. To complete the query, we join LibraryEmployeeType table containing hourly_wage and type. The query was designed as follows:

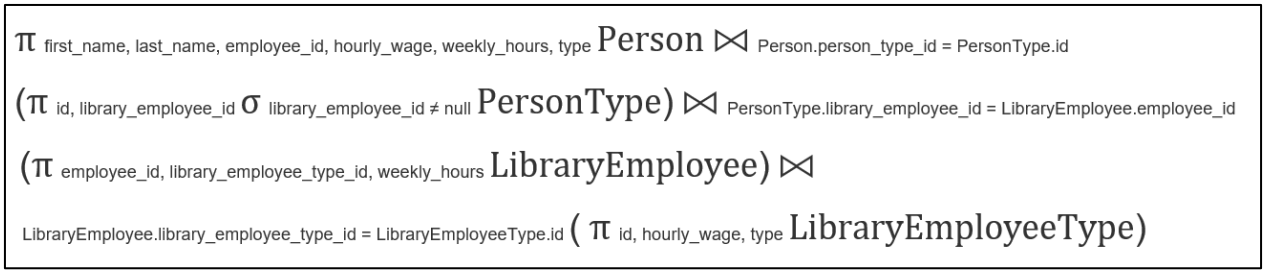


Figure 69 Algebraic form of the personal information of employees and their wages

After the implementation:

```
SELECT P.[first_name], P.[last_name], LE.[employee_id], LET.[hourly_wage], LE.[weekly_hours], LET.[type] FROM Information_Basic.Person AS P
INNER JOIN (SELECT [id], [library_employee_id] FROM Information_Basic.PersonType WHERE library_employee_id IS NOT NULL) PT
ON p.person_type_id = PT.id
INNER JOIN (Select [employee_id], [library_employee_type_id], [weekly_hours] FROM Information_Sensitive.LibraryEmployee) LE
ON PT.library_employee_id = LE.employee_id
INNER JOIN (SELECT [id], [hourly_wage], [type] FROM Information_Sensitive.LibraryEmployeeType) LET
ON LE.library_employee_type_id = LET.id
```

Figure 70 The personal information of employees and their wages query in SQL

In the first step we declare all the fields we want to display as the result. Next step for the SQL server to perform is the inner join of two tables Person and PersonType. The joining keys on which the join is performed are foreign key person_type_id form Person table and id form PesonType. After the data is retrieved the next step is to join next table which is LibraryEmployee. It is possible by comparing PersonType.library_employee_id with LibraryEmployee.employee_id. To have all the fields available to display the third LibraryEmployeeType table must be joined. It is possible by joining the tables on the id and library_employee_type_id fields. All the performed joins are inner joins as the data we want to retrieve must be consistent and visualize the connected records from several tables. Another important part of this query is a "*WHERE*" clause in the select statement from the PersonType table. By stating that library_employee_id must not be "*NULL*" we ensure the result contains only the records of people working in the library.

Performance

In the case of this query the execution tree looks as follows:

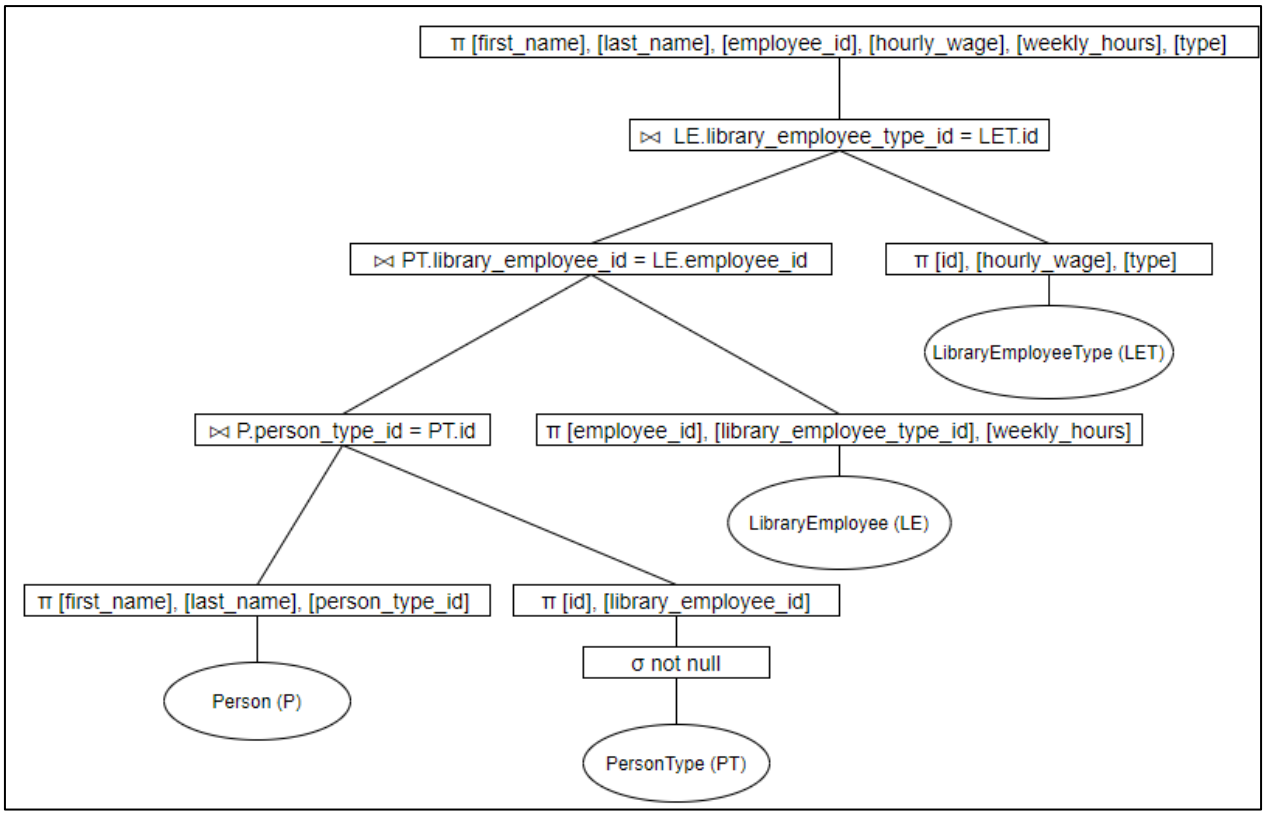


Figure 71 Execution plan of the personal information of employees and their wages query

After the examination and discussion, we decided to put the non-clustered index on the field library_employee_id in PersonType table as the "*WHERE*" operation must be performed on that data.

Before the indexes were implemented, we made a control run. The part we are focused on is the Clustered Index Scan on PersonType table. The result shows:

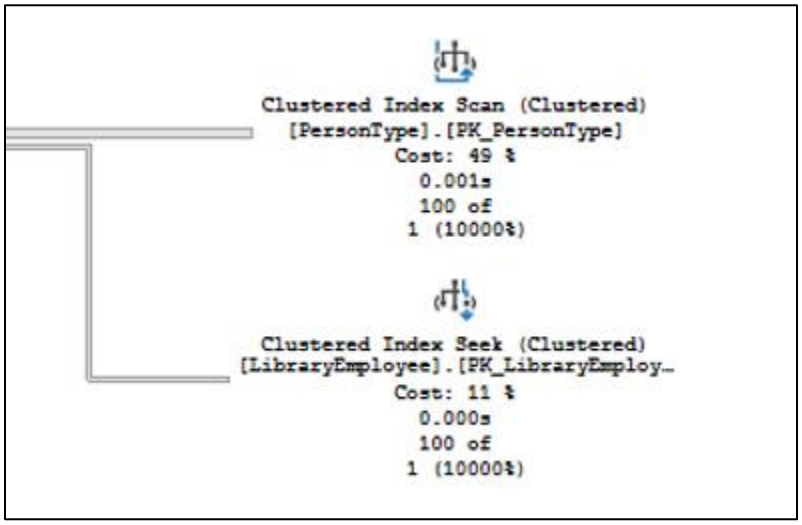


Figure 72 Part of the execution plan of the personal information of employees and their wages query in SQL

The query had been run again after the non-clustered index implementation:

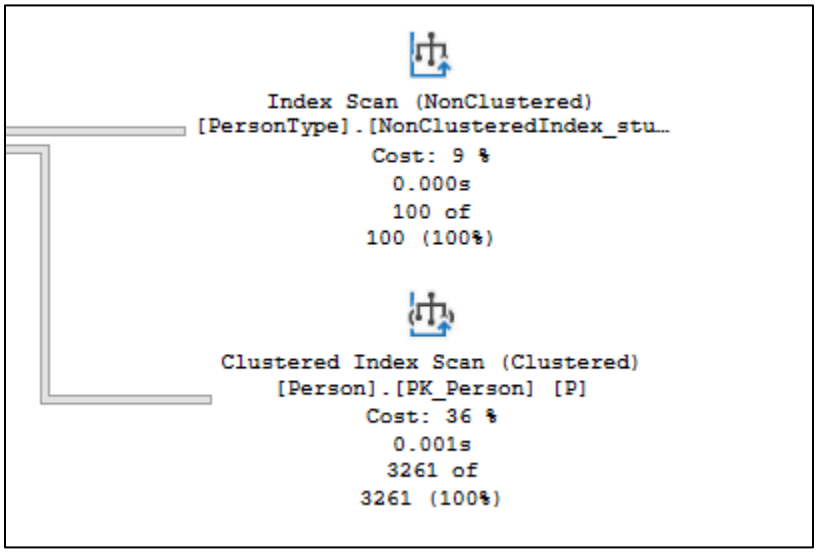


Figure 73 Part of the execution plan of the personal information of employees and their wages query in SQL with non-clustered index

As it is visible after the implementation the time of execution was lowered from 0.001s to 0.000s. However, more significant difference can be spotted by examining the cost of the operation. From 49% of consumed resources it was lowered to 9% which is a huge improvement. We expect to achieve more significant results regarding time differences of execution when the data will grow.

As a further examination of the query execution we noticed that in this case SQL server chose to use Nested Loops and Hash Match kinds of join.

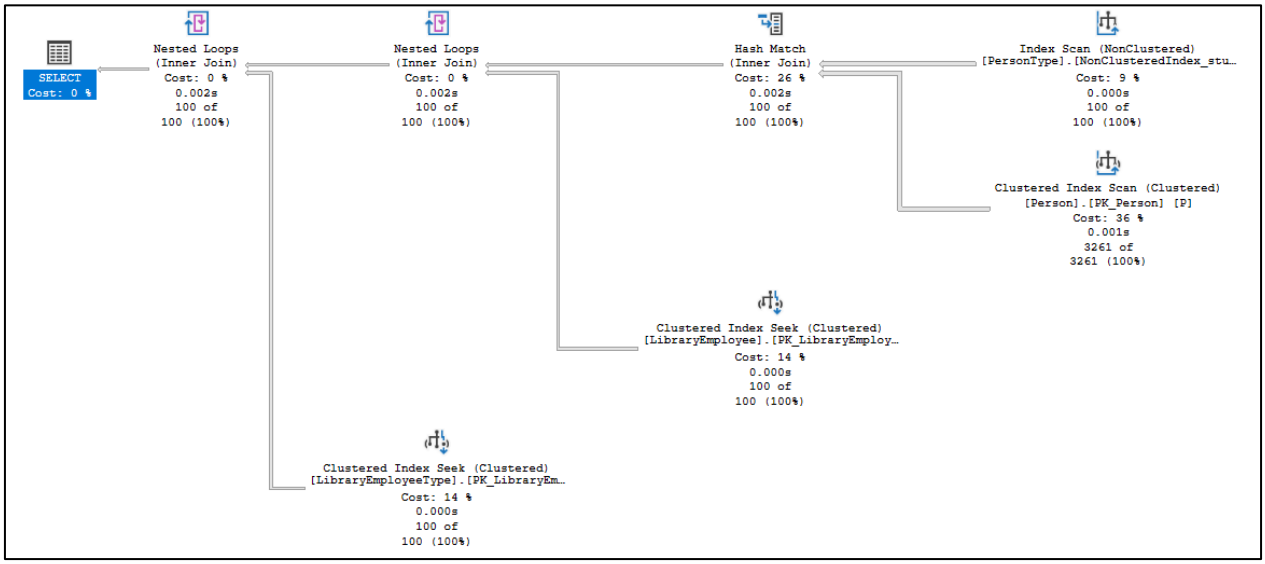


Figure 74 Execution plan of the personal information of employees and their wages query in SQL

Considering the definitions of different join kinds, the hash match is applied while dealing with big tables. In our case the both PersonType and Person tables contain a big number of records. Furthermore, another

important factor that made the server to choose this type is the index position. On both tables adequate indexes are put. Next operation after the result of hash match is created is nested loop. Going by the definition it also applies perfectly in that case. The result of joining PersonType and Person on the condition that retrieved record contains information about the employee is small. Therefore, looping does not consume many resources and runs relatively fast. Reflecting on the choice of Nested Loop we also deducted that any other kind of join is not possible to use in this case. The reason is the result of the hash match as it does not contain suitable indexes neither it is sorted. The same reasons apply to the last join operation which combines the results from earlier with LibraryEmployeeType table.

3.9.4 Information about people that are late with book returning

Another important information for every library is a record of borrowers that did not return the books on time. The data we are interested in is the personal information of registered students the borrow date and the expected return date together with the information to identify the book.

	ssn	first_name	last_name	card_number_id	borrow_date	isbn	estimated_return_date	status
1	011-53-8195	Quentin	Pollard	1045946125	2020-05-12	2852297228944	2020-05-07	0
2	011-53-8195	Quentin	Pollard	1045946125	2020-05-12	2852297228944	2020-05-07	0

Figure 75 Result of the query

Considering all the requirements we designed the query:

$$\pi_{\text{ssn, first\_name, last\_name, borrow\_date, isbn, estimated\_return\_date}} \text{BookBorrow} \bowtie_{\text{BookBorrow.book\_catalog\_id = BookCatalog.id}} (\pi_{\text{isbn, id}} \text{BookCatalog}) \bowtie_{\text{BookBorrow.book\_return\_id = BookReturn.id}} (\pi_{\text{estimated\_return\_date, status, id}} \sigma_{\text{estimated\_return\_date > current\_date and status = 0}} \text{BookReturn}) \bowtie_{\text{BookBorrow.ssn = Person.ssn}} (\pi_{\text{first\_name, ssn, last\_name}} \text{Person})$$

Figure 76 Algebraic form of the query

After the implementation:

```
SELECT P.[ssn], P.[first_name], P.[last_name],BB.[borrow_date],BC.[isbn], BR.[estimated_return_date] FROM Loan_Activities.BookBorrow BB
INNER JOIN (SELECT [isbn], [id] from Book.BookCatalog) BC ON BB.book_catalog_id= BC.id
INNER JOIN (SELECT [estimated_return_date],[status],[id] FROM Loan_Activities.BookReturn
WHERE DATEDIFF(DAY,[estimated_return_date],Convert(DATE,GETDATE()))>0 AND [status]=0) BR ON BB.book_return_id = BR.id
INNER JOIN (SELECT [first_name], [ssn], [last_name] from Information_Basic.Person) P ON BB.ssn = P.ssn
```

Figure 77 Query form in SQL

To make the date relevant for the librarians several fields must be displayed. Therefore, the first row specifies all of them. The first table we select data from is BookBorrow and its field borrow_date. It also contains the book_catalog_id field which also is a foreign key. Thanks to that we are able to retrieve more information about the book such as isbn by performing an inner join with the table BookCatalog on book_catalog_id and id fields. To be able to distinguish the information about the loans on the books that has been returned and on those which has not, we need to retrieve information about the estimated return date and the status. Therefore, the next step is to select data from the BookReturn table, filtered accordingly to the requirements. The method we use is DATEDIFF which is natively supported by the SQL server. As the argument we put "*DAY*" which indicates on which time interval we are interested in. After that the estimated_return_date and a todays date in the right format. As a result, we receive a number which says how many days a specific book has been kept after the deadline by the borrower. After the "*AND*" clause another condition is specified which filters the results such that only not returned books are returned. After all that data is retrieved, we can inner join the BookReturn and BookBorrow tables on id and book_return_id.

The last set of information the query returns is stored in the Person table. To retrieve the right results, we inner join the BookBorrow table which contains SSN foreign key with SSN primary key from Person.

Performance
The execution tree created after the query examination looks as follows:

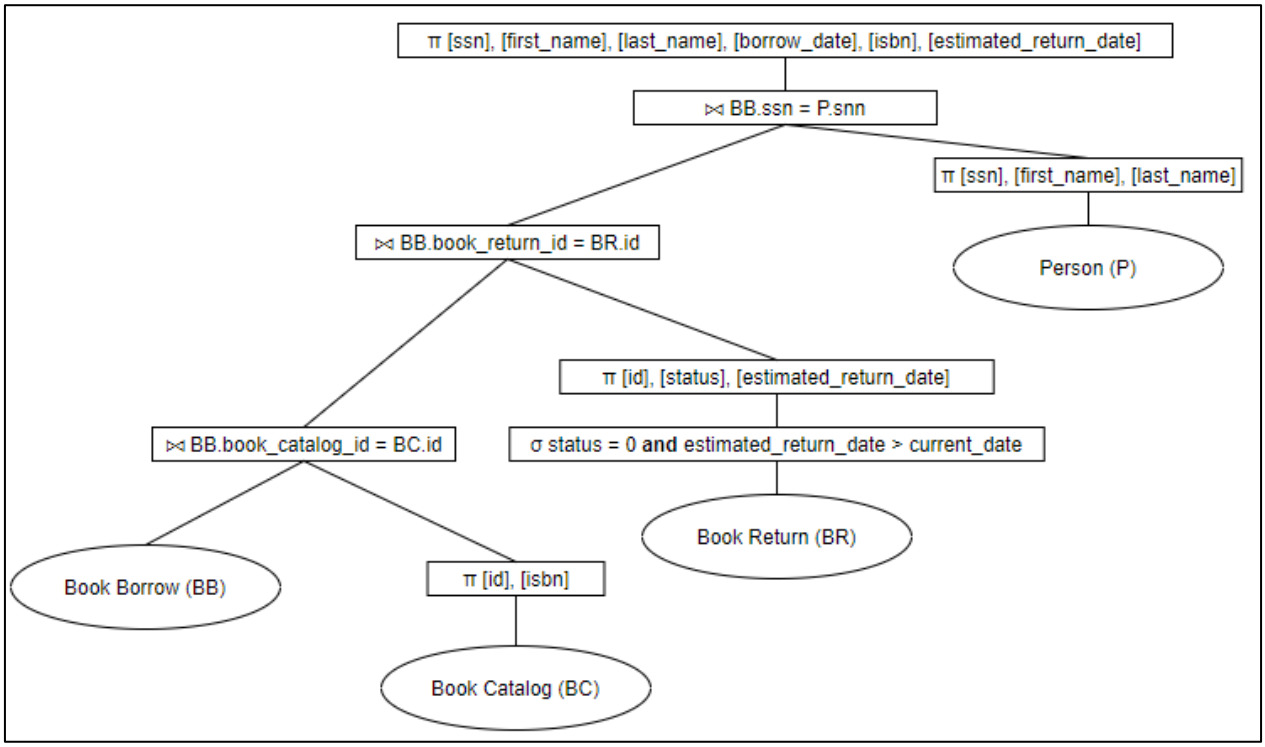


Figure 78 Execution plan of the query

As the diagram shows the query contain only one "*WHERE*" clause and therefore we decided to implement the non-clustered index on the estimated_return_date field in the BookReturn table. The control run result:

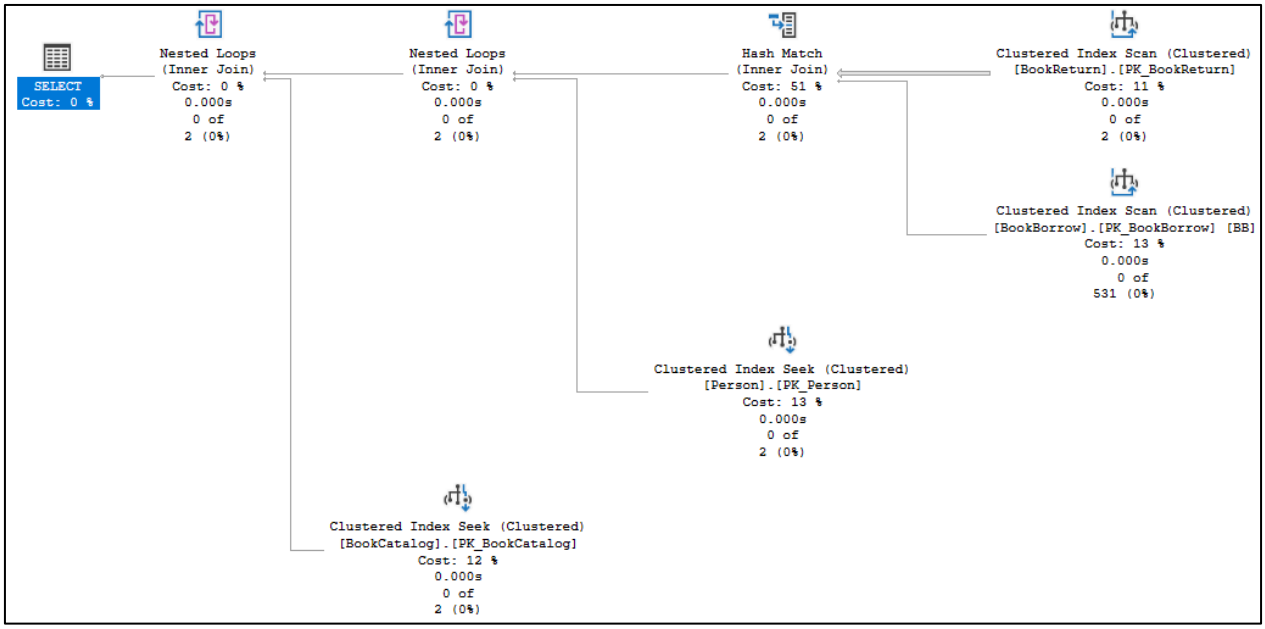


Figure 79 Execution plan of the query in SQL server

With the index implemented we noticed a difference in the execution:

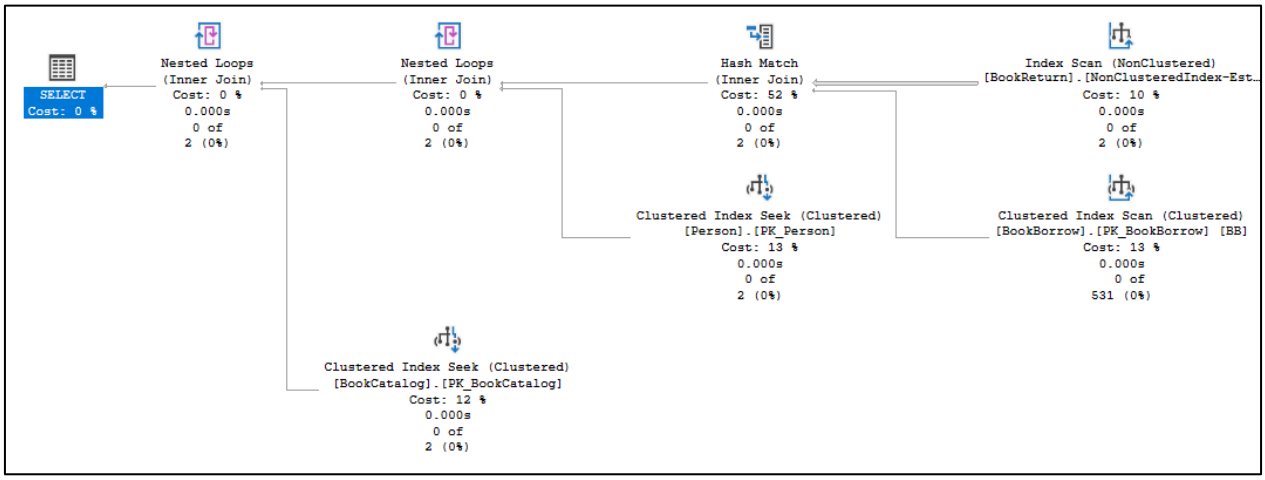


Figure 80 Execution plan of the query in SQL server with non-clustered index

As the figure shows the Clustered Index Scan has changed to Index Scan by the non-clustered index. The change is insignificant because the execution time is not visible, and the cost lowered only by one percent. However, the BookReturn table together with BookBorrow in the future will contain the biggest amounts of data. Therefore, we expect the difference to be more noticeable and it is a good practice to implement all the improvements up front. Regarding the types of joins SQL server used in this case the first executed is hash match. The size of both tables is the reason which corresponds to the definition. The next operations are nested loops. The result of the first join is a small set of data and therefore nested loops do not consume many CPU cycles to execute.

3.9.5 Books available to borrow

The last query we decided to design will find its use for every user of the library. The goal is to retrieve the information about all the available book copies.

id	isbn	book_condition_id	is_deleted
25	1949741454368	3	0
27	1949741454368	3	0
28	1949741454368	3	0
30	1615227248684	1	0
31	1615227248684	1	0
33	1615227248684	1	0

Figure 81 Result of the query.

To retrieve data in the form, show on the figure three tables must be joined together. The first step is to query the BookCatalog table. All the fields displayed are part of this table. However, in the next steps tables BookBorrow and BookReturn must be joined to filter the result according to the requirements.

From the beginning in the designing phase our group discussed two different solutions. One of the possible ways of solving this problem is to implement both and based on the performance chose more suitable method of implementation. The first idea to test was based on two joins. The designed query:

Another method to implement this query is to filter the BookReturn and BookBorrow tables using correct clauses and use "*IN*" or "*NOT IN*" syntaxes. This allows to return the rows that exist or do not exist in a specific result set.

$$\pi_{\text{id, isbn, book\_condition\_id, is\_deleted}} \sigma_{\text{BookBorrow.book\_catalog\_id = null and BookCatalog.book\_condition\_id} \neq 2} \text{BookCatalog} \bowtie_{\text{BookCatalog.id = BookBorrow.book\_catalog\_id}} ((\text{BookBorrow}) \bowtie_{\text{BookBorrow.book\_return\_id = BookReturn.id}} (\pi_{\text{id}} \sigma_{\text{status} = 0} \text{BookReturn}))$$

Figure 82 Algebraic form of the query with joins

The implementations of both methods look as follows:

```
SELECT BC.id, BC.isbn,BC.book_condition_id, BC.is_deleted FROM [Book].BookCatalog AS BC LEFT JOIN(
(SELECT * FROM Loan_Activities.BookBorrow) AS BB JOIN
(SELECT ID FROM Loan_Activities.BookReturn WHERE Status = 0) AS BR
ON BB.book_return_id = BR.ID) ON BC.ID = BB.book_catalog_id WHERE BB.book_catalog_id is NULL
AND BC.book_condition_id <> 2
```

Figure 83 First version of the query in SQL

As the figure shows the first step is to specify the fields intended to display. Next step is to perform "*LEFT*" join to be able to retrieve all the records form the left table. The joining is performed on the BookCatalog table and the result of the expression from the brackets. The first SELECT returns all the data from BookBorrow table so that it is possible to join it with BookBorrow on the BookBorrow.book_return_id and BookReturn.ID fields. It retrieves the data about all the borrowed books. Going further by having the result of previous selects the "*LEFT JOIN*" expression is finished by pairing the records. The fields we use are BookCatalog.ID and BookBorrow.book_catalog_id. The last step is to filter the result using "*WHERE*" clause. As the result set still contains borrowed books the expression retrieves only records where book_catalog_id is null returning only available copies. Additionally, some of the books are not available for borrowing because of their condition. Therefore, the filtering also includes expression returning records where the condition is different than "2".

The implementation of the second method looks as follows:

```
SELECT * FROM [Book].BookCatalog
WHERE id NOT IN(SELECT book_catalog_id FROM Loan_Activities.BookBorrow
WHERE book_return_id IN (SELECT ID FROM Loan_Activities.BookReturn WHERE Status = 0)) AND
book_condition_id <> 2
```

Figure 84 Second version of the query in SQL

The table we start with is BookCatalog. In the next step we start to filter the result by executing "*WHERE*" clauses. It is executed against the result of SELECT statement from the brackets using "*NOT IN*" expression. The result set contains the data from two tables BookBorrow and BookReturn. Going down into the execution tree we start with the BookReturn.ID field and "WHERE" filtering the status to be "0". Next step is to retrieve book_catalog_id from the BookBorrow table which is also present in the result from previous expression. The highest operation is selection of BookCatalog records which are not in the result from the brackets and also where the conditions is different than "2".

Performance

In this case as we have two possible solutions and therefore, we will compare them in the manner of their performance. As a first step we tried to find right candidates to put the non-clustered indexes. Equipped with the knowledge from the previous examples we were able to quickly identify the right implementation. After the control run the execution plans look as follows:

Using joins

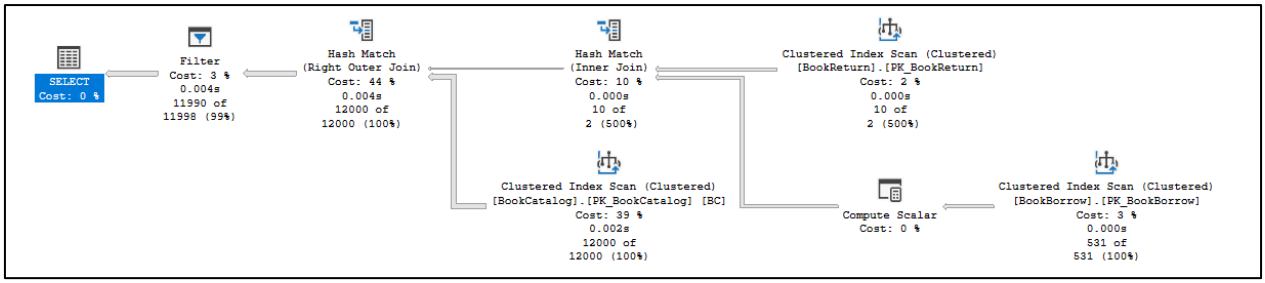


Figure 85 Query execution plan in SQL with using joins

Using in and not in

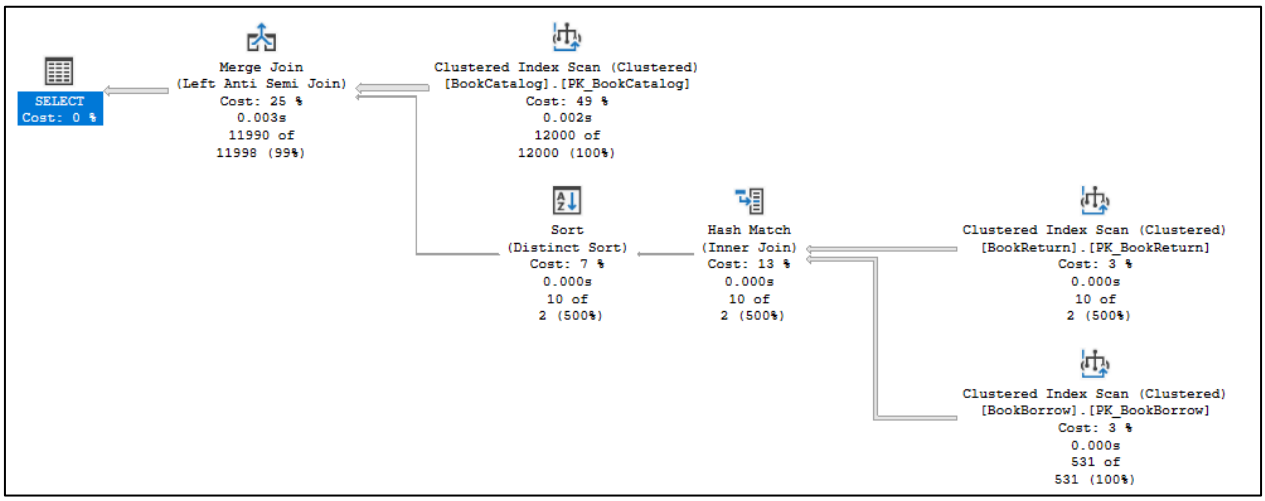


Figure 86 Query execution plan in SQL with using "IN"/ "NOT IN"

The fields on which the non-clustered index would work the best are Status in the BookReturn table and book_return_id in the BookBorrow table.

Using joins

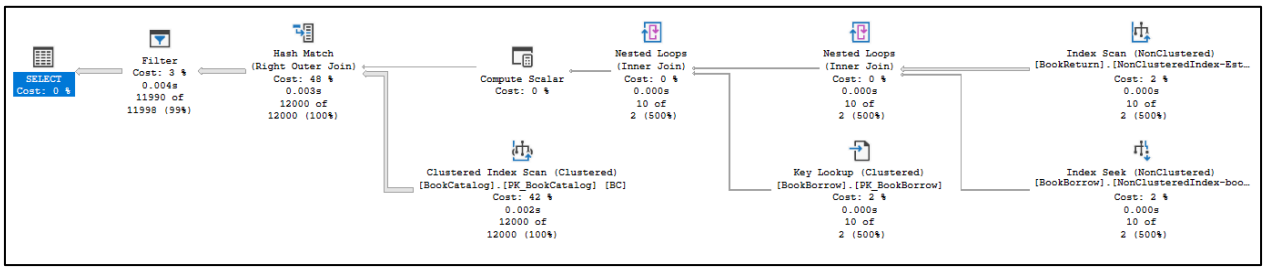


Figure 87 Query execution plan in SQL with using joins and non-clustered index.

Using in and not in

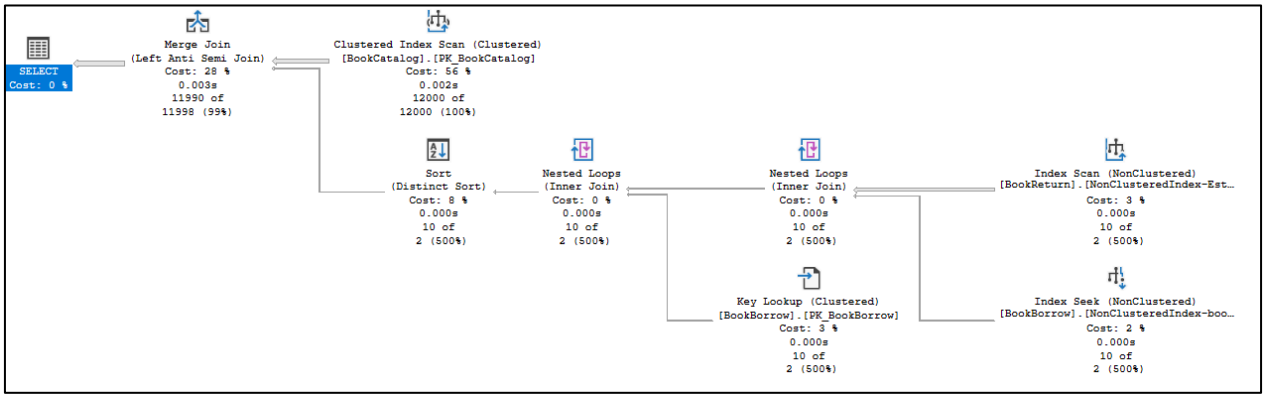


Figure 88 Query execution plan in SQL with using "IN"/"NOT IN" and non-clustered index

As the figures show the execution plan changed significantly. By looking on both cases Index Scans changed to Index Seeks. However, these are not the only changes. The execution plan after the implementation of non-clustered index changed in the manner of different kinds of joins SQL server chose. Considering the first case which is created using joins the hash match changed with nested loops. Similar update is visible in the query which uses in and not in. The difference in time and cost is barely visible however, in the future it could be more significant.

After both queries were examined and tested our group decided to stay with the second solution which uses “IN” and “NOT IN” expressions. The performance tests showed its slight advantage and it is easier to read and understand.

All the designed queries were explicitly chosen so that they will be relevant for the software created in the later part of the project. Therefore, we planned to create views using them in an already optimized form. However, it will be described in the further part of this report.

3.10 Views

Depending on the requirements different data must be fetched from the database. In some cases, to retrieve all the information the user is interested in, a complex query must be executed. In the case of our project the situation is similar. Moreover, they are the most commonly sent requests by the users. While developing a software we can avoid sending heavy requests to the database by using database programming. The SQL server can store complex queries which join many tables together in a view.

View is a kind of virtual table which also consists of rows and columns. It is also possible to specify whether we want all the fields from selected tables to be displayed or only some of them. During our project as one of our tasks was to design and implement five different queries and optimize them, we decided to make use out of them. Therefore, they were planned to be useful in our software and in the further part we wanted to create views using the code already implemented. Moreover, for practice and to implement the code faster we created additional views ending up with a total number of seven.

While creating the view the only requirements are name and the query we want to save. Based on the example of the "loan activities history" query from section 3.9.1, the SQL code looks as follows:

```
CREATE VIEW [dbo].[BookReturningHistory] AS
SELECT BB.[ssn], BB.[borrow_date], BC.[isbn], BR.[estimated_return_date], BR.[returned_date], BR.[payment], BR.[status]
FROM dbo.BookBorrow BB
INNER JOIN (SELECT [id], [estimated_return_date], [returned_date], [payment], [status] FROM dbo.BookReturn) BR ON BR.id = BB.book_return_id
INNER JOIN (SELECT [id], [isbn] FROM [Book].BookCatalog) BC ON BB.book_catalog_id = BC.id
```

Figure 89 BookReturningHistory view creation script

As the figure shows after the "CREATE VIEW" clause we must specify the name of the view and optionally the schema we want to assign it to. Next expression is "AS" and after that the query we want the view to execute. Those principles apply to every view we created and therefore we will not describe every one of them explicitly.

Most of our views were created using the five queries from section 3.9 such as:

3.10.1 AvailableBooks

```
SELECT TOP (1000) [id]
      ,[isbn]
      ,[book_condition_id]
      ,[is_deleted]
  FROM [Giorgia_Tech_Library].[dbo].[AvailableBooks]
```

Results | Messages

id	isbn	book_condition_id	is_deleted
9	2296321897998	3	0
11	2296321897998	1	0
15	2296321897998	3	0
16	2296321897998	1	0

Figure 90 Available books view result

3.10.2 BookReturningHistory

```
SELECT TOP (1000) [ssn]
      ,[borrow_date]
      ,[isbn]
      ,[estimated_return_date]
      ,[returned_date]
      ,[payment]
      ,[status]
  FROM [Giorgia_Tech_Library].[dbo].[BookReturningHistory]
```

Results | Messages

ssn	borrow_date	isbn	estimated_return_date	returned_date	payment	status
321-30-9427	2019-04-19	1615227248684	2020-04-10	2020-04-08	0	1
459-56-4777	2019-08-25	3355933265239	2019-09-15	2019-09-06	0	1
684-38-6511	2019-07-10	1149733462322	2019-07-31	2019-07-21	0	1
274-83-6067	2019-11-01	1237825496438	2019-11-22	2019-11-13	0	1
490-05-6145	2019-07-02	1718251372392	2019-07-23	2019-07-14	0	1

Figure 91 BookReturningHistory view result

However, as mentioned above we also created additional views to make the development easier and this section will focus more on those instead of repeating already described code.

One of the most important functionalities every library should have is to keep track of the condition of the specific book copies. Therefore, we created a view which returns all the destroyed positions. Consequently, the library has insight and eventually order additional copies. The query which creates the view looks as follows:

```
CREATE VIEW [dbo].[DestroyedBooks] AS
SELECT * FROM Book.BookCatalog WHERE book_condition_id = 2
```

Figure 92 DestroyedBooks view creation query

The data it retrieves:

id	isbn	book_condition_id	is_deleted
4	2815812115447	2	0
6	2815812115447	2	0
8	2815812115447	2	0
12	2296321897998	2	0
13	2296321897998	2	0
14	2296321897998	2	0
26	1949741454368	2	0

Figure 93 DestroyedBooks view result

After the view was created our group discussed the possible performance improvements we could implement. The query contains one where clause on book_condition_id in the BookCatalog table and therefore we decided to create non-clustered index.

Although the index was implemented there was no performance improvement. Furthermore, the execution plan visible in the Management Studio did not change either.

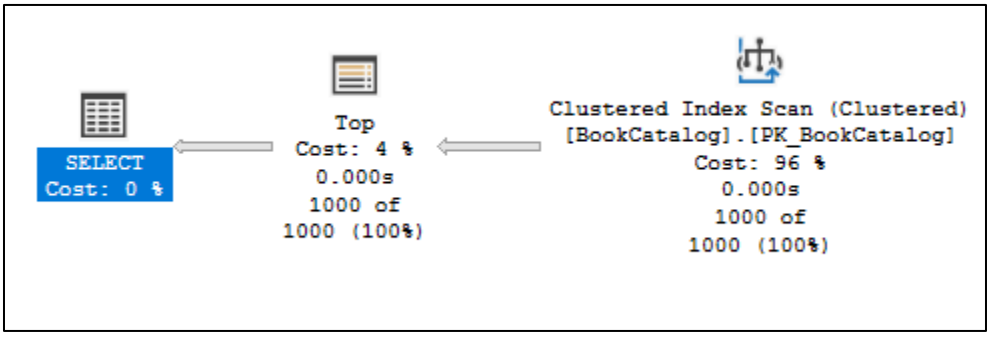


Figure 94 Execution plan for DestroyedBooks view

Next crucial information for the library is a set of records with people who are late with returning a book. By implementing this functionality, the library can create statistics and keep track of students who commonly return books after the deadline. The query code looks as follows:

```
CREATE VIEW [dbo].[PeopleWhoDidNotReturnedBooksYet] AS
SELECT P.[ssn], P.[first_name], P.[last_name], P.[card_number_id],BB.[borrow_date],BC.[isbn], BR.[estimated_return_date], BR.[status]
FROM [dbo].BookBorrow BB
INNER JOIN (SELECT [isbn], [id] from [dbo].BookCatalog) BC ON BB.book_catalog_id= BC.id
INNER JOIN (SELECT [estimated_return_date],[status],[id] FROM [dbo].BookReturn WHERE DATEDIFF(DAY,[estimated_return_date],Convert(DATE,GETDATE()))>0
AND [status]=0) BR ON BB.book_return_id = BR.id
INNER JOIN (SELECT [first_name], [ssn], [last_name], [card_number_id] from [dbo].Person) P ON BB.ssn = P.ssn
```

Figure 95 PeopleWhoDidNotReturnedBooksYet view creation query

The result set it produces:

ssn	first_name	last_name	card_number_id	borrow_date	isbn	estimated_return_date	status
011-53-8195	Quentin	Pollard	1045946125	2020-05-12	2852297228944	2020-05-07	0
011-53-8195	Quentin	Pollard	1045946125	2020-05-12	2852297228944	2020-05-07	0

Figure 96 PeopleWhoDidNotReturnedBooksYet view result

In this case all the improvements regarding the performance are were already made in the previous part while creating and optimizing five queries. The indexes were put on estimated_return_date, status, and book_borrow_date. Therefore, we will not describe this part briefly. The execution plan of the view:

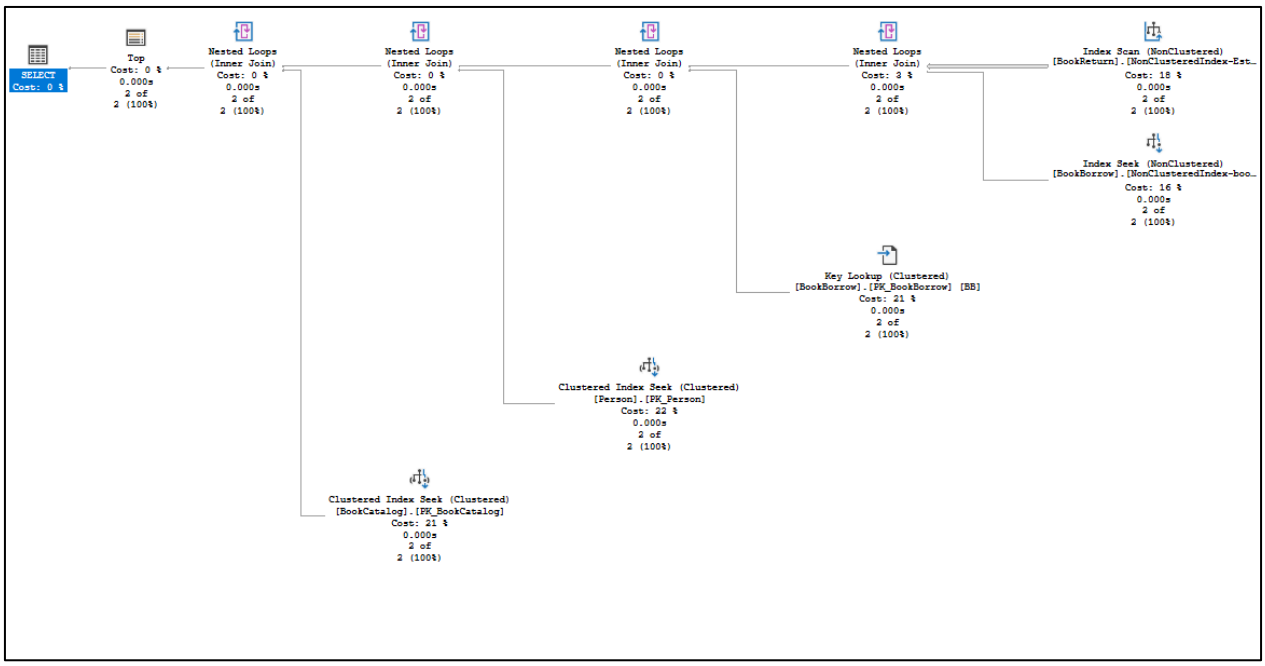


Figure 97 Execution plan for PeopleWhoDidNotReturnedBooksYet view

The views we created made the development easier and faster. They are convenient way of saving the queries. Moreover, the developers can make a good use out of the views regarding the database security, but it will be described in the further part of this report.

3.11 Stored procedures

SQL server provides the developers with one feature more which allows to move some of the logic from the software to the database itself. Using SQL, a stored procedure can be created. It is a prepared code that can be saved so that it can be reused. Therefore, if there exist a query which that is used often the developer can save it and later just call it to execute. Another important feature of stored procedures is possibility of passing parameters. Based on the value of the input stored procedure can act differently.

In every library only some of the operations are executed the most often. As an example of those functionalities we can name borrowing and returning a book or retrieving a history of actions for a specific user. In our system the situation is indifferent and therefore we decided to implement some of them as stored procedures.

3.11.1 Return Book and Change Status

The first functionality our group chose is returning a specific book and changing the status of the loan. It will be often used by every user. The information required for this operation are the catalogue id of a book and the card number of a user. The raw query of this operation looks as follows:

```
UPDATE [dbo].[BookReturn] SET [status] = 1 FROM [dbo].[BookReturn] AS BR JOIN
(SELECT book_return_id, book_catalog_id FROM [dbo].[BookBorrow] AS BB where BB.book_catalog_id = SOMEID AND BB.ssn =
(SELECT ssn FROM [dbo].[Person] AS P JOIN (SELECT number FROM [dbo].[Card] WHERE number = SOMECARDNUMBER) AS C
ON P.card_number_id = C.number)) AS BBJ ON BR.id = BBJ.book_return_id WHERE BR.[status] = 0
```

Figure 98 ReturnBookAndChangeStatus raw query

It is an “UPDATE” query with a couple of joins which allows to locate right user and the book. The first step is to find a specific student by the card number. Therefore, a join operation on Card and Person table must be made. After the card number is found in the Card table it is matched with Person.card_number_id

foreign key. The result from the person table that we are interested in is ssn so that the SELECT operation on BookBorrow table can be completed. Besides the ssn it also requires a book_catalog_id which identifies the specific book copy. The result of those operations is a BookBorrow record which contains book_return_id. Having that we can easily perform an update on the BookReturn table by joining it with BookReturn on the foreign key book_return_id and primary key id.

Creation of a stored procedure is not much different from the query which creates a view and looks as follows:

```
CREATE PROCEDURE [dbo].[ReturnBookAndChangeStatus]
@theCatalog int,
@cardNumber int
AS UPDATE [dbo].[BookReturn] SET [status] = 1 FROM [dbo].[BookReturn] AS BR JOIN
(SELECT book_return_id, book_catalog_id FROM [dbo].[BookBorrow] AS BB where BB.book_catalog_id = @theCatalog AND BB.ssn =
(SELECT ssn FROM [dbo].[Person] AS P JOIN (SELECT number FROM [dbo].[Card] WHERE number = @cardNumber) AS C
ON P.card_number_id = C.number)) AS BBJ ON BR.id = BBJ.book_return_id WHERE BR.[status] = 0
```

Figure 99 ReturnBookAndChangeStatus procedure creation query

As the figure shows the shows the first line states that the sorted procedure is to be created and the name is specified. After that we can initiate a number of parameters starting with "@" and after space we must decide what type of variable the field is going to store. To reference the parameter in the query afterwards only the name with "@" is required.

After the sorted procedure is created it can be executed using the SQL code.

```
EXEC dbo.ReturnBookAndChangeStatus @theCatalog = 3, @cardNumber = 1563212
```

Figure 100 Execution query for the procedure

Expression "EXEC" calls a specific procedure and after the name is given the parameters must be assigned a value.

3.11.2 Specific User's Books to Return

One of the most important information for every user is to know what books one must return. Thanks to that data students can keep track of the positions they borrowed. As the best solution for identifying a specific person is by card number this is the only parameter required for this stored procedure.

```
SELECT* FROM dbo.Book WHERE isbn IN (
    SELECT isbn FROM dbo.BookCatalog WHERE id IN(
    SELECT book_catalog_id FROM dbo.Person JOIN(
    SELECT * FROM dbo.BookBorrow WHERE book_return_id IN (
    SELECT id FROM dbo.BookReturn WHERE [status] = 0)) AS
    Borrow ON Borrow.ssn = dbo.Person.ssn WHERE card_number_id= SOMECARDNUMBER)
    )
```

Figure 101 SpecificUserBooksToReturn raw query

Although, the description sounds rather simple, the query requires several tables to be joined and filtered. The first steps are to retrieve all the BookReturn records which are not returned yet. Having the id, we can filter the BookBorrow table using "IN" expression finding only the records we are interested in. The result set from the BookBorrow table contains foreign key ssn which can be joined with Person table. Additionally, as Person table holds card_number_id as a foreign key by using "WHERE" clause, we can easily filter the results and retrieve only the ones corresponding to the specific user. Having the data filtered, by filtering of BookCatalog table we can return specific isbn of the books which has been borrowed. In the last step the

Book table is filtered and all the information such as title, author or publication date are retrieved and returned to the user. The process of creation is identical the previously described stored procedure.

```
CREATE PROCEDURE [dbo].[SpecificUserBooksToReturn]
    @cardNumber int
AS
    SELECT* FROM dbo.Book WHERE isbn IN (
    SELECT isbn FROM dbo.BookCatalog WHERE id IN(
    SELECT book_catalog_id FROM dbo.Person JOIN(
    SELECT * FROM dbo.BookBorrow WHERE book_return_id IN (
    SELECT id FROM dbo.BookReturn WHERE [status] = 0)) AS
    Borrow ON Borrow.ssn = dbo.Person.ssn WHERE card_number_id= @cardNumber)
    )
```

Figure 102 SpecificUserBooksToReturn procedure creation query

In this case only one parameter @cardNumber is required and it holds and int value.

3.11.3 Other Stored Procedures

Our team to practice and to make the future development of the software easier came up with some more operations which can be saved as stored procedures. The process of creation is always identical and therefore we will not describe every procedure explicitly. However, it is worth to show our ideas.

One of the procedures returns personal information together with wages and weekly hours of the employees based on their first and last names as parameters.

```
CREATE PROCEDURE [dbo].[SearchEmployee]
@specifiedFirstName varchar(30),
@specifiedLastName varchar(30)
AS
SELECT P.[ssn], LE.[employee_id], P.[first_name], P.[last_name],   LET.[hourly_wage], LE.[weekly_hours], LET.[type] FROM dbo.Person AS P
INNER JOIN (SELECT [id], [library_employee_id] FROM dbo.PersonType WHERE library_employee_id IS NOT NULL) PT ON p.person_type_id = PT.id
INNER JOIN (Select [employee_id], [library_employee_type_id], [weekly_hours] FROM dbo.LibraryEmployee) LE ON PT.library_employee_id = LE.employee_id
INNER JOIN (SELECT [id], [hourly_wage], [type] FROM dbo.LibraryEmployeeType) LET ON LE.library_employee_type_id = LET.id WHERE P.[first_name] = @specifiedFirstName
AND P.[last_name] = @specifiedLastName
```

Figure 103 SearchEmployee procedure creation query

This query will find its use in the administration.

Many of the users would like to see full history of their actions. Consequently, as it will be common request, we decided to implement a stored procedure.

```
CREATE PROCEDURE dbo.[SpecifiedUserHistory]
@theSsn char(11)
AS
SELECT BB.[ssn], BB.[borrow_date], BC.[isbn], BR.[estimated_return_date], BR.[returned_date], BR.[payment], BR.[status]
FROM dbo.BookBorrow BB
INNER JOIN (SELECT [id], [estimated_return_date], [returned_date], [payment], [status] FROM dbo.BookReturn) BR ON BR.id = BB.book_return_id
INNER JOIN (SELECT [id], [isbn] FROM [dbo.BookCatalog) BC ON BB.book_catalog_id = BC.id WHERE ssn = @theSsn
```

Figure 104 SpecifiedUserHistory procedure creation query

As the figure shows the search is performed based on the ssn and not the card number of a user. The reason for that is practice and an idea to connect two procedures together. Going further, we implemented an operation to retrieve a specific user by the card number:

```
CREATE PROCEDURE dbo.[FindPersonByCardNumber]
@theCardNumber int AS
SELECT * FROM dbo.Person AS P JOIN (SELECT * FROM dbo.[Card] WHERE number = @theCardNumber) AS C ON P.card_number_id = C.number
```

Figure 105 FindPersonByCardNumber procedure creation query

This stored procedure will be commonly used in different places in our software as every identification starts with the card number of a user.

The stored procedures are very useful when it comes to complex queries which are often called. They can make the development easier and save effort of the developers. Equipped with this knowledge in the future we will be able to compare the solutions of implementing features in the software or the database and choose the most suitable one for a given requirement.

3.12 Security

Security in database is an important part of real-industry problem solving. Even the greatest database system is not worth a lot when no control over who and what can be done to the database without our knowledge. Therefore, these conditions create a need for some kind of layers of functionality or confidentiality for specific data.

In case of our project we decided to base our security rules on logins, users, and schemas.

Login – used for connecting to the Database server. It can be used for example to define certain database access for specific login. Here we can determine which databases instance does specific user related to that login has access to.

User - used for authorization to specific SQL Database. Can be assign schemas to control permissions to specific tables within the database instance.

Schema – collection of logical structures that such as tables, views, stored procedures, triggers etc. With predefined operations restrictions.

*****One login can relate to many users within a database server but within one specific database there is 1 to 1 relation between a **Login** and a **User**

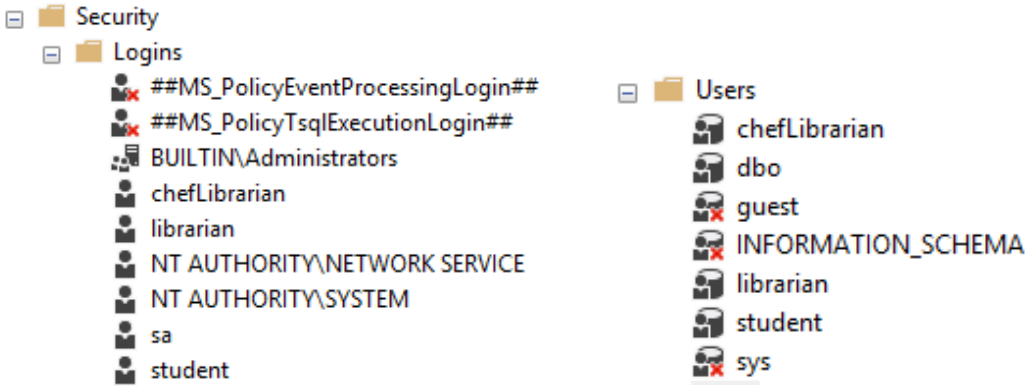
The way how we handled login selection to the database is out of the scope of this part of the report but in the end, we end up having 3 kind of users that login accordingly to who they are. Then for every specific user we assigned relevant to his desired permissions schemas.

We started by creating 3 logins to the DB server and selected that each of them should have access to both to the production and testing database.

Users mapped to this login:

Map	Database	User
☑	Giorgia_Tech_Library	librarian
☑	Giorgia_Tech_Library_Test	librarian

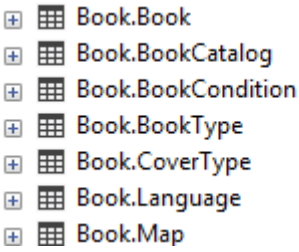
That instantly added 3 users of the same name to our databases instance. That is *student, librarian, chefLibrarian.*



Now we divided our database tables, views procedures etc. into functionality layers based on business domain and that was the base for our schemas.

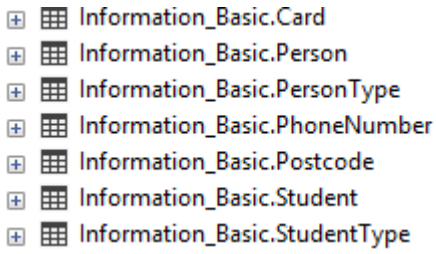
We added following schemas into our database:

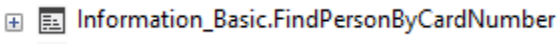
Book – here we have all the tables related with books - condition, language, number of copies etc.



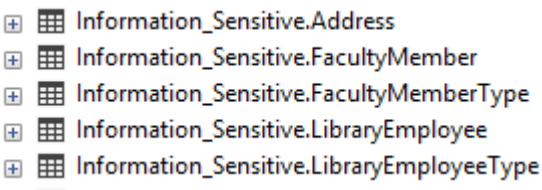
Loan_Activities – tables and stored procedures related with borrowing and returning the books.

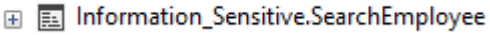
Information_Basic – all the information tables about students (excluding adress) and one stored procedure for finding a person based on card number .

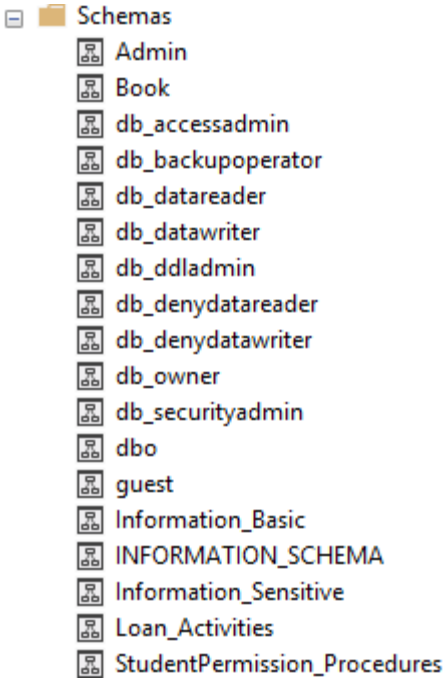




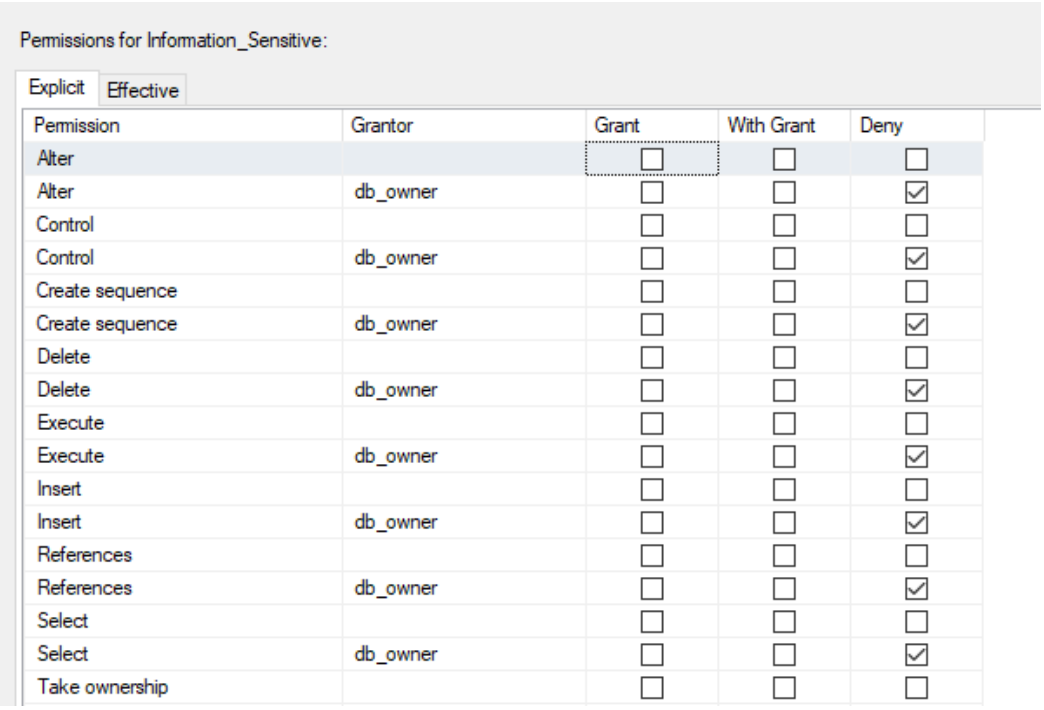
Information_Sensitive – peoples` addresses, information about employees and lecturers + stored procedure for searching an Employee.







Then for each of the users we limit the activities he can take within the specific table. As an example, let us take the *student* user and the schema *Information_Sensitive* we **DENY** him from taking any of the listed activities.



3.13 Conclusions

We managed to adapt knowledge gained during database course into this project. It has been a good on hands experience with database development, maintenance, optimization, and security. It also allowed us to improve our skills both in business understanding as well as addressing the requirements. Throughout the process we have been exploring different solutions and discussing its relevance. Decision we undertook were based on our skills, knowledge and resources(time). In the end we managed to from requirements analysis, get to the working database. Use of EER diagrams allowed us to communicate with stakeholders in a technical language that they know and understand. Rules on how to create relational diagram and then create the database allowed us to present the customer with high quality solution for assessed problem. Then we investigated different solutions about the security and programmability and implemented the ones that we found relevant from business domain perspective and the ones that we manged to handle programmability.

In future if our time budget were bigger, we would definitely spend more time on exploring alternative ways of handling the security and functionality such as user roles. We would also like to dive deeper into the issues related with exposing the data to users. We plan to implement the statistics using MongoDB, that we did not manage this time due to the time management.

The entire group have contributed to the database we have implemented. We shared a lot of knowledge and it was definitely beneficial to work with many people and learn about their competence.

3.14 References

1. https://www.w3schools.com/sql/sql_join.asp (1355)
2. https://en.wikipedia.org/wiki/First_normal_form
3. https://www.studytonight.com/dbms/database-normalization.php
4. https://www.tutorialspoint.com/Partial-Dependency-in-DBMS
5. https://www.mssqltips.com/sqlservertutorial/9133/sql-server-nonclustered-indexes/
6. https://www.mssqltips.com/sqlservertip/2115/understanding-sql-server-physical-joins/
7. https://tomyrhymond.wordpress.com/2011/10/01/nested-loop-hash-and-merge-joins/
8. https://ucn.instructure.com/courses/23353/files/folder/Module%2006?preview=1126092
9. https://beginnersbook.com/2015/04/transitive-dependency-in-dbms/
10. https://www.geeksforgeeks.org/first-normal-form-1nf/
11. https://www.lifewire.com/transitive-dependency-1019760
12. http://www.gitta.info/LogicModelin/en/html/DataConsiten_Norm3NF.html
13. https://www.studytonight.com/dbms/boyce-codd-normal-form.php
14. https://www.sqlshack.com/delete-cascade-and-update-cascade-in-sql-server-foreign-key/
15. https://www.mssqltips.com/sqlservertutorial/9133/sql-server-nonclustered-indexes/
16. https://www.w3schools.com/sql/sql_join.asp
17. https://www.mssqltips.com/sqlservertip/2115/understanding-sql-server-physical-joins/
18. https://tomyrhymond.wordpress.com/2011/10/01/nested-loop-hash-and-merge-joins/
19. https://ucn.instructure.com/courses/23353/files/folder/Module%2006?preview=1126092
20. https://www.funtrivia.com/askft/Question7781.html
21. https://en.wikipedia.org/wiki/List_of_long_place_names

TEST

1. Preface

The motivation for this report was 1st semester Software Development top-up program project. It is created by a team of four students who together create a team. The report is a description of our achievements during the development cycle we conducted. The contents include the process of deciding on the right test strategy and approach. Furthermore, based on the results our team documents the process of testing including creation of test scenarios and implementation of tests. Additionally, we are focused on modern tools and practices and therefore usage of few chosen is described. The purpose of this report is to document the steps we undertook while completing the tasks we were given during the project.

2. Abstract

The report briefly describes pre-project work, planning, implementing, and optimizing test cycle. The purpose of this report is to document our process of designing the strategy of delivering quality software and the actions we took to implement, and use given solutions. We described different testing strategies and approaches. Additionally, based on the requirements we were given we documented the process of choosing the right method and reflected on our choice. Going further we described the practices used while following a certain process model. Following the risk driven testing methodology we designed and implemented a set of tests. To conclude the project, we reflect on our achievements and we evaluate the process.

3. Tests

3.1 Introduction

Testing is a process that issues both static and dynamic activities related with the planning, execution and evaluation of the software. It that attempts to reduce number of errors in the program by reducing the number of undiscovered defects remaining in the software. As the combination of possible tests even for a simple module/unit is practically infinite software engineering testing require **test strategy** to determine generic feasible solution that process based on skills, time and resources of the project. Then the project requires **test approach** that is specific implementation of test strategy for given project determines key elements of the testing such as whether it is going to be **risk** or **requirements** based or what kind of external **standards/processes** are going to be followed.

Testing has its 7 principles:

1. Testing shows only the presence of defects – testing does not prove correctness.
2. Exhaustive testing is impossible – it is simply not possible to check everything.
3. Early testing - it is better to test early; test the requirements when they are not completely implemented rather than changing design and requirements for already complemented software.
4. Defects cluster – small number of modules is most likely going to be responsible for major part of problems. It is redefinition of Pareto Principle – “around 80% of problems are found in 20% of modules”.
5. Pesticide paradox - the more you test the software the more immune it becomes to the tests.
6. Testing is context dependent - each software project requires unique approach that might indeed share general testing strategy but will be different than any other software project when it comes to testing approach.
7. Absence of error is a fallacy - even software without bugs might be unusable due to wrong implementation of requirements. If the test is invalid but its result is valid the software ends up working incorrectly.

3.1.1 Quality in software

In software engineering quality refers the degree which certain information system meets its requirements and customer expectations. Therefore, quality is divided into 2 main parts that both reflect the quality of the software but in a different way:

Internal quality – Is the quality of the product. It determines how good is the process that we are developing software with. How well designed is the software technically and visually? How good is the code being written by developers? What is the level of testing and reviews?

External quality – Is the quality of what the customer sees/uses. It addresses issues like the performance, maintenance, UX. More generally speaking when speaking of external quality, it is high when the product does actually solve the problem.

3.1.2 Various test examples

Unit test also called component test – is checking whether certain module/component behaves as intended. They are constructed by developers, usually parallel with the construction of the software. Sometimes (In test driven development) the tests are implemented first. This kind of testing is executed by programmers.

Acceptance test – is customer involved black box testing that checks whether the programmers have met the requirements. It is divided into stages: First the acceptance tests should be carried out on the supplier's side and then on the customer`s side. This activity is part of hand over procedure.

Installation test– you perform it typically when the system is installed to make sure that it is working correctly in its new environment.

Live test – these kinds of test usually address components that were not possible to test in the lab/ development environment.

System test – verifies whether the system meets both functional and non-functional requirements. It includes relevant procedures (e.g. installation) and it requires controlled environment.

Operational acceptance test – takes place after the release has been finished (usually from 3 to 6 months). This test is performed to make sure if the system fulfils the operational contract.

3.1.3 Black-box vs white-box testing

Black-box testing – this method does not require the developer to know the details about the software under testing. Certain test is provided input and the expected output. Test validates whether the output is correct but does not go into details about how it has been achieved.

White-box testing – this method does require the developer to know details about the software under testing. Internal knowledge about the program is used to design the tests. An appropriate input is selected by the tester and then he validates all the steps that the system has to undertake in order to achieve as well as the output itself.

3.1.4 Test types

Among the tests we can distinguish different test types that all address different part of user/system requirements and that is:

Functional testing – testing functional requirements of the system/specification. Validates only the test result - comparing input with output without examining the process (black box testing).

Non-functional testing – testing non-functional requirements such as performance, usability, reliability, restart time etc.

Structural testing – focuses on structure of the code rather its functionality. It validates the way the system achieves specific tasks, rather than its result.

Confirmation testing – re-testing software that previously failed.

Regression testing - re-testing software that already worked as expected.

3.2 The process model

The process model is a set of features for a wide range of application that provides complete and adequate definition of any which it is applied to.

It consists of 6 parts: figure 3.

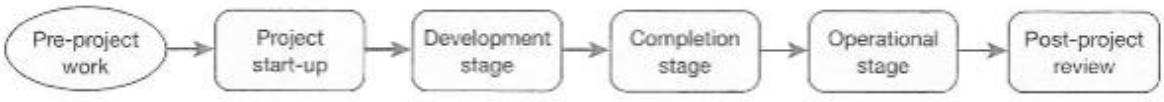


Figure 4. Process model

1. **Pre-project work -** is an initial stage during which customer after evaluation of tenders selects specific supplier and they reach an agreement. During this phase initial scope and objectives of the project are determined.

2. **Project start-up/Initiation stage –** is a stage that covers project`s foundations for future work. This stage should not be underestimated as it has great influence on projects success/failure in future. At this point of the project all parties want to start the actual work already. Nonetheless it is in best interest of both the customer and the supplier to carry out this process with great care. Even comprehensive contracts crafted in pre-project work stage are unable to cover all the issues that appear prior to development stage. Therefore, it is up to the project manager to resolve all the remaining uncertainties in the project start-up so that the development stage is as efficient as possible. It is useful to divide the project start-up into: **What, Why, Who, how and when** headers**.** The management during this stage have following documents as tools to answer listed previously questions:
   - *Project initiation document (PID)* – is a major document of project`s start-up stage that encapsulates its main decisions. It is produced by project manager afterwards approved by project sponsor. Which in regular case would be a person that is responsible for whole project from customers side. In internal organisation project the sponsor could for example be a person from receiving department that is responsible for cooperating with the delivering department(supplier). PID format my vary but one might use OSCAR (objectives, scope, constraints, authority, resources) technique to create it.
   - *Project Plan* – is a document used to both guide both project execution and project control. Its main parts are to planning assumptions and decisions.
   - *Quality Plan* – is a document used for defining attributes of quality strategy such as: development approaches, quality criteria to be applied to each of the products, quality manuals and development approaches.
   - *Risk management plan* – is a document defining the risk management process. It is a set of procedures of defining, assessing, and controlling threats of the project.
   - *Project organisation structure* – is a document that represents unique characteristics and design of the organisational structure. Its main purpose is to avoid any confusion on communication level of the process.
   - *Project administrative procedures* - system of rules that govern the procedures for managing a project. These procedures are meant to establish efficiency, consistency, responsibility, and accountability.

3. **Development stage –** is a stage where majority of supplier`s work is carried out. These activities are fully on supplier side and it is the supplier`s manager that takes responsibility for these actions. During this phase customer performs so called factory acceptance which is essentially test performed on development side of the software that happens before releasing the product. Even though each project is unique and independent unit we can determine some common parts of development stage and these are:
   - *Requirements definition* – is a part of the development stage during which previously defined general requirements and expectation become detailed. Here the desired goal is to have all the requirements captured in a complete, consistent, and clear format so that it is easy to tell whether certain objective has been met or not. This allows us to make sure that we are following customer`s recommendations.
   - *Design* – is part of the development stage that address the issue of how the requirements of the project are to be met. This addressed both the logical and the physical of design of the system. All design is supposed to have logical link to the requirements meaning that they are supposed to be derived from the project's objectives.
   - *Implementation* – is part of the development stage during which both the programming and unit testing take place. During this phase, the customer has the least influence on the project. Nevertheless, the project`s director will still monitor whole process and take relevant actions to steer the supplier into correct directions when necessary. At the end of this phase it is expected to have a complete set of tested with respect to the specification modules.
   - *Integration and testing* – part of the development stage during which particular modules from implementation phase are tested together. This attempts to spot any kind of data format inconsistency or flaws in the communication process of the modules.
   - *System testing* – is part of the development stage during which entire system is validated vs design documents and requirements specification. Another task would be to check the system according to the acceptance criteria so that the system being delivered.

4. **Completion stage –** is a stage that starts after the information system has been complete. It must have been thoroughly tested and all of the errors resolved. We must make sure that all of the requirements are fulfilled. The customer receives finished product and starts examining whether the program meets the agreed upon specification. The stage finishes when customer accepts the software. Steps that are relevant to the completion stage are:
   - *Delivery to the customer* – is a formal handover during which customers receives all the elements of the software including documentation. Details about the handover are usually addressed in project plan.
   - *Training and familiarization* – a process of teaching the customer`s staff/users on how to use the software. It`s recommended to start this activity earlier in the process. As an example, factory testing done by users during development stage of the software ease the onboarding process.
   - *User acceptance testing* – is a process during which customers applies series of testing both on the system and the documentation. Acceptance testing can be divided into 4 categories: Functionality (functionality defined in the requirements of the system), Performance (e.g. response time, users number, no. of transaction per time unit), Interface testing (checks intercommunication with other systems), Environmental testing(power consumption, noise etc.)

- *Acceptance by customer* – is a step that takes place when all the testing mentioned previously is accepted (or is running with a low enough fault percentage). This is activity might require the customer to sign acceptance certificate or similar document.
- *System commissioning* – is a step during which the software is being mounted on its final environment and number of **site acceptance tests** are being repeated using real data while being connected to other systems. Capacity and stress testing require the system to run with its regular/ near regular traffic.
- *Final customer takeover* – this activity officially ends the projects. In some cases, the customer would take over system with faulty functionality and agree with the supplier to fix it in specific time range. For the project director it is also time to reflect on the entire project and find thing that the company might do better/benefit in future from.

5. **Operational stage –** is a stage that starts after the live running begins. It`s usually not part of the project unless support/guarantee has been agreed beforehand. Most of the information system require minor fixes, enhancements. This is due to the fact that with time passage the business requirements change. These changes should be implemented into the software using the same techniques as in development stage. The products of Operation stage are:
   - *Fixes* – this might address new errors that happen after the users get their hands-on entire software or previous errors that were supposed to be solved but due to the new scenario, they start reoccurring.
   - *Enhancements* **–** as users get to know the information system better and better, they start to realise its limits. As a result, they begin to determine both useful and redundant features. That leads to certain usability of the software that could come in handy for the users but is currently not part of the information system.

6. **Post-Project review –** is an activity that takes place around 6 months after the implementation. Its main purpose is to check whether the business objectives of the system have been met. Unlike to previous checks it focuses more on the benefit that the company have.

3.3 Pre-project work

During our pre-project work with The Georgia Tech Library (GTL) we wanted to accomplish 3 things:

- Create initial view of the project
- Define the objectives of the project
- Define the scope and type of agreement for the project

3.3.1 Initial view of the project

We started by initial analysis of case study document GTL have provided us with. At first the group briefly went through it and we have talked about our observations, described the problem each with our own words. From there we brainstormed a little and decided to write down all relevant information.

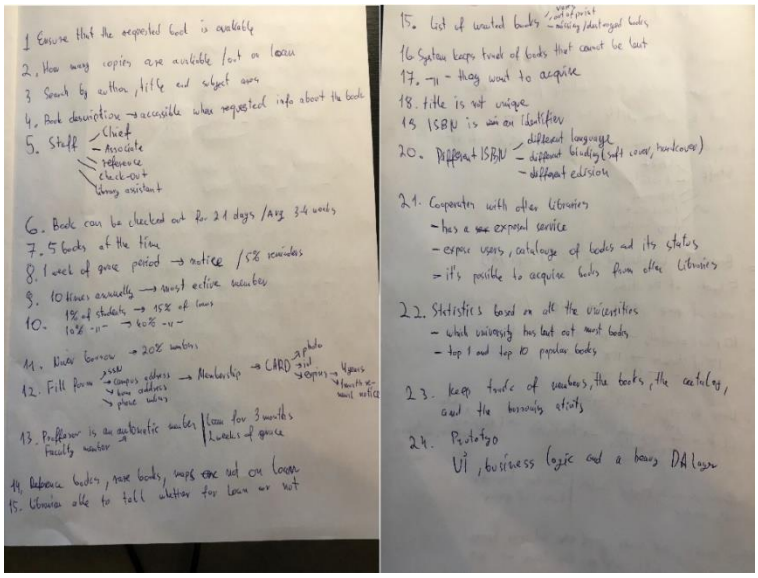
First, we wrote down general information about the GTL: the number of members, how many titles do they have, and how many of all the volumes are out on loan averagely at any time.

- 16,000 members
- 100,000 titles
- 250,000 volumes in total
- 25,000 volumes out on loan any time

3.3.2 The objectives of the project

From there we decided that we need to have a basic overview of what are the objectives for GTL software. To broaden the perspective that the group was investigating this task we came up with following process.

- **Each** of the group members wrote down relevant issues that our information system is supposed to address. As a result of that we came with list of bullet points.
- Seconds step was to one by one go through all of the points and discuss their relevance. Then formed **one** list for the whole project.
- In the end we selected all the points and made handwritten draft of all the objectives we found relevant to the Georgia Tech Library case.
- As the next step we decided to in more detailed and documented way write down a description based on the handwritten document and create a table with name and description of the objective. That document was based on the previously mentioned list but was not a 1 to 1 reflection as we were still discussing and decided to merge some points and introduce small changes. The final version of our thought process is represented in the table below.



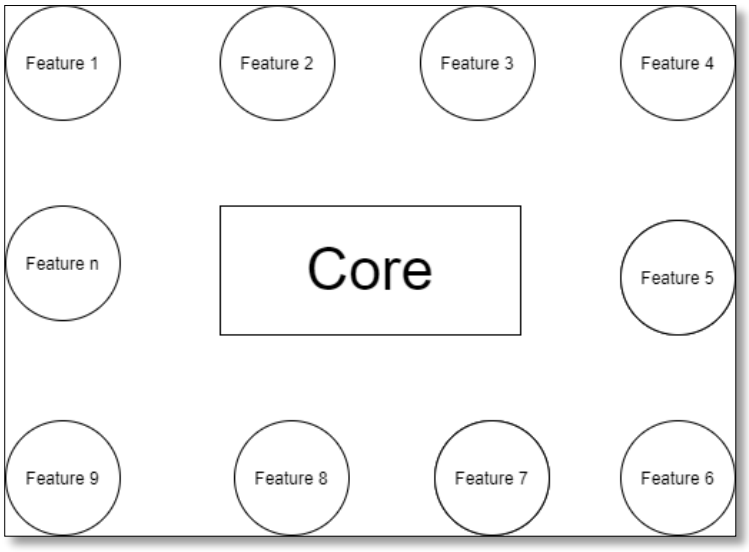
Name of the objective	Detailed description
1. Add and return book	This is the most basic functionality of the library where users are able to borrow and return the books and this functionality is supposed to be part of the system.
2. Ensure that the reuqested book is avaliable	When a person sees a book as avaliable it is actually in the library ready to be lent out from the library. *Optionally implement reserving system.
3. How many copies are available/out on loan	The number of available and out on loan books must be reliable. We need to make sure that we do not count copies that are not eligible for borrowing (rare books, reference books, maps and destroyed books). Moreover we need to trace which books are being lent and which are not and change their statused when users return or borrow them.
4. Search by author title and subject area	It is possible to display the all the books in the catalog and filter them by three categories: Author, title, and subject area

5. Accessible book description when information about a book is requested	Whenever a user wants to borrow a book, while finding a desired position, its description should be displayed. Consequently, the user can read the desription and get insight of the book content.
6. Library Staff	The library staff has multiple ranks/levels. There is at the top Chief librarian , then associate librarians, reference librarians, check-out staff and library assistant. When implementing employees in our system it is desired to have some kind of division implemented as we expect to have various privilege leveles based on a position.
7. Book can be checked out for different period depending on perons who borrows it.	The Georgia Tech Library policy allows students to borrow a certain book for 21 days. However faculty members's loan period is extended to three months.
8. Five books at the time	Any user of the library system can borrowed up to 5 books at the time.
9. Grace period	We will trace when users return their books. If the returning date is withing a week after the estimated returning date, then students do not pay any fee. In case when faculty members are late with returning a position to a library on time, then they do not have to pay a fee if the returning date is withing two weeks after the estimated returning date.
10. Basic Information about users	Most active users are the ones who borrow at least 10 times per year Top 1% of students make 15% of loans Top 10% students make 40% of loans Around 20 % of users never borrow books
11. Professors and Faculty members are instant users	When a new faculty member joins, their data is pulled automaticly from their records and they become users of the library.
12. Not all materials should be avaliable for loan	Some of the materials in the library should not be borrowed by users. These are reference and rare books as well as maps. Librarian should see in the system whether the resource he is currently looking at is unavilable for loans.
13. Form for account creation	Whenever there is a request for creating a new user of the GTL library, a special form, should be filled. The form contains SSN, campus address, home address and a phone number. Having the form filled, it is possible for library employee to issue a card for a new user. The created card consists of a user's photo, has unique number and expiration date. Each card is valid for four year. One month before the expiration day, a renewal notficiation should be sent to a student. The card usage is required to borrow and return books in the library.
14. The library has list of wanted books	There is a record of books/resources the library wants to get/buy, that includes rares, out of prints, and missing/destroyed materials. The system is supposed to keep track of this record.
15. Book uniqueness	Each book is identified by its ISBN number. Titles of stored books are not unique. Therefore, there might be a case when two different books have the same title. Moreover there can be multiple copies of a specific book. It is essential to ditiquish them from each other, because some of copies may not be eligible for borrowing. Two same book (mening it is from the same author and has the same title), still might have different ISBN if either: 1. Languague of a book is different, 2. Cover of a book is different, 3. Edition of a book is different
16. Integration with other libraries	The GTL library cooperates with other libraries. The libraries have a service that they use to communicate with each other. They expose the users, catalogue of the books and their status. Also it is possible to acquire some of the books through this service. It is desired that our system can integrate into their endpoints as well as expose its own.

17. Statistics	The GTL also wants to expose some of the statistics based on all of the libraries: - Average loan time - Which university lent our most books Top 1/Top 10 most popular books
18. Prototype resposibility	The initial Georiga Tech Library software should keep track of members, book, catalog of books and currently borrowed and available units. There is an API implemented and a client which exposes its endpoints. API's structure is a repository patter. It consists of three layers: controllers, services and respositories. 1. Repository layer is responsible for retrieving data from a database, 2. Service layer is responsible for handing bussines logic, 3. Controller layer is responsible for exposing endpoints for its user and its function is some kind of interface for applications that use the API. Client is web-based and it sends requests to the API and awaits for its responses. Depending on the response result, an appropriate instructions are dispayed to a user.

3.3.3 Define the scope and type of agreement for the project

As we had initialized the objectives of the project we could have decided on its type. Because of the fact that, a lot of the requirements have been incorporated into the study case document, we were able to create quite detailed list of requirements for our program. We have concluded that we are dealing with fixed budget project. We planned our developmnet stage to start on the April the 13th and finish on May 22nd. This gave us 5 sprints to acomplish what we want. Then from 22nd to 28th of May we would focus on the closure of the project. As we were not completely confident with a project of such nature and expected several spikes to occur, we decided that realistic approach to do this project would be to define some fundamental businesss requirements relevant to GTL (the core) and implement it. Then if we had more time we could repeat this proccess and again: select from remaing issue the ones that are the most crucial and relevant from business perspective and implement those.



3.3.4 Selection of the project's core

We decided to select the core for the project by first defining what is the main functionality of the library. While implementing these features we will consider all the constraints from the objectives list table above. Giving an example, it could be a grace period (no. 9), loan tome (no. 7) or books' ISBN number uniqueness (no. 15). The outlined core of the GTL system would be:

- Main functionality of the library is to provide its users with knowledge. That means that there are users and they should be able to find, borrow and return books that match their needs or can be beneficial to them. Objectives that points to the feature: [1, 2, 4, 7, 9, 12, 13, 15, 18]
- As for the library employees' point of view, they should be given a basic tool to control what is happening in the library. It is tracing its users (students), number of available and on-loan books as well as the catalogue of all books in the library. Following objectives will be fulfilled while writing the implementation: [3, 4, 12, 13, 15, 18]

Having our initial objectives as well as scope for the project concludes the pre-project work stage and allows us to move further and start using those two things to build up a project initiation document in the project start-up phase.

3.4 Project start-up

The work we have done in previous stage is going to be the basics for what we are about to do in this phase. As a team we were aware that the results of the project start-up are going to influence the project during its whole lifecycle. Therefore, no rush has been taken during this activity.

3.4.1 Products of the project start-up

We decided to combine Project, Quality and Risk management plan into one document. This decision has been based on the *fifth edition of Project management for Information Systems by James Cadle and Donald Yeates.* It is explained that depending on the size and the scope of the project, these three documents may be combined in one. Furthermore, they may also address issues related with the project organisational structure as well as project administrative processes. Unfortunately, because of the project characteristics, the organisational structure and project administrative processes of GTL is out of the scope. This leaves us with 2 tasks to fulfil:

- Capture all the main decisions of the project – create the Project Initiation Document (PID)
- Prepare document describing our Project, Quality, Risk plans and administrative procedures (PQRP)

3.4.2 Project Initiation Document

We used OSCAR formula, in order to create our Project Initiation Document. That is objectives, scope, constrains, authorities and resources. Both objectives and scope has been already mentioned in the pre-project work. Nonetheless we will reflect back again to them.

Project Initiation Document

1. Objectives
   The business objective of the project is to encourage more people to use the Georgia Tech Library. Delivering the software would make the usage of the library easier and probably bring more people to borrow books. Consequently, the library might count on some state funds and develop its infrastructure. The project objective is to implement a software that would modernize the work of the Georgia Tech Library.
2. Scope

Boundaries - because of the project's nature it is not possible to state which departments are included into the scope. Nonetheless, we can call ourselves an IT department.

Activities - After a long discussion, the team together as a project director decided that, we are creating a prototype and some assumptions have to be done in order to deliver working prototype for the GTL with respect to the time budget.

The assumptions we made was about the users' management:

> We are not going to support creation of new users. Instead the database will be filled with a number of people and created accounts with cards for them. If we were in real industry and Project Director would have decided differently, we would respect his decision and prioritize this activity. That was decided unanimously, if we had on board regular Project Director who would have decided differently, we would of course respect it.

The scope of the project is to deliver a product that allows its users to borrow and return books. At the same time fulfilling all the constraints from the defined objectives that the scoped feature would point to. Despite the functionality, the system would also provide an opportunity for employees to trace library's system users, see a book catalogue as well as book copies that are available and out on loan. Implementation of displaying statistics, description of books, adding new users and librarians, modifying already existing materials and users will not be undertaken during the lifespan of the project.

Deliverables- The project team will be working on an implementation of an API that will provide the defined objectives as well as on a client web-application which will be used by users to make actions. These two components will be produced and handed over to the customer at the end of the project.

3. Constraints

In earlier part of this document we described the way we are going to implement that software. That is: we are going to prioritize some of the requirements in order to fulfil the core functionality of the library and then decide what to do next based on the time we have left. Therefore, the method we are looking for is rather adaptive than prescriptive. Also, it seemed like we are going to have some kind of iterations in our project. Meaning that the planning, development testing and deployment is going to be repeated for each of the increments we do. This is why we decided to select Agile as our software development method and inherit from parts of the SCRUM in terms of lifecycles. In order to keep our development formal, we have decided to write down group contract to formalize structure of our work, responsibilities, work time and software we are going to use.

GENERAL RULES ABOUT WORK AND WORKSPACE
1. Each of the members is expected to work 6-8 hours every weekday
2. Concerned for our health and following government restrictions we are not going to meet in person through the whole project or until else decided.
3. Online meetings will be held using Microsoft Teams, where related documents such as current tasks are going to be shared.
4. Each of the members is responsible to at the beginning of the day check for any updates in those documents.
WORK TIME
1. We are going to let the developers be as flexible as possible, but some ground rules should be determined: • Our meeting time is from 12.00-13.00 every day. Meaning that it is expected of each of the team members to be available for online meeting, pair programming, general help etc. during that period. • If more time is required during particular day a meeting should be planned at least day before.
ABSENCE
1. If any of the members has to take care of personal issues and therefore, they are unavailable in particular day they should announce it beforehand at least day before using general chat on Microsoft Teams

To address some of the time constrains: This issue has already been assessed during the pre-project work. When determining the scope of the project we have also decided on reasonable date for the end of Completion Stage. The system should be finished and tested approx. 1 week before the deadline which is 28th of May. We chose the last Friday before that date for the closure that is 22nd of May. The remaining six days we will spend on polishing the project and eventual enhancements/fixes. This also allows us to have a kind of buffer in case of significant delay/ accident.

4. Authorities

We determined 3 roles that are involved in the development of information system for Georgia Tech library. Two of them were authorities and the remain one was for the team. The following administrative roles are:

1. Customer Project Manager also called *Project Director* who is our main stakeholder and is responsible for accepting the project and solving business related conflicts. In our case it was unable to have that manager onboard from GTL. Therefore, it was decided that this manager`s responsibilities will be divided equally among the team members throughout entire project. We believe that this might be the fuel for discussion and is going to be beneficial.

2. *Supplier Project Manager* who takes care of technical issues related with development of the system. As none of us had previous experience in such position from real industry we decided to have a rotating position for Supplier Project manager. His responsibilities were to solve technical arguments have decision-making vote in given week.

3. *The team* who is responsible for developing the project and implementing the software.

5. Resources

We are four-people development team that works on delivering the product due to agreed date. The equipment that we possess is not much. They are personal computer of each of us. Because of the current situation in the word, we are forced to work remotely through platforms that allows us to. Our database is being hosted on the DigitalOcean cloud. Assumptions about delivery dates and

number of features that will be implemented were also made by looking at the resource that we have. Some of the project's development team members are working part-time. Consequently, an overall efficiency of the team goes down.

3.4.3 Project, quality, and risk management plan

We decided to concatenate project, quality, and risk management plans into one. The plan for the project is to hand-in a tested and functional software with iteration-based features implemented due to the deadline. We are going to work agile. Therefore, there is a high flexibility of choosing the order of the features' implementation. Nonetheless we have decided to firstly focus on most risky components of the software that is going to be delivered. We are going to define the most risk-based features by creating a risk analysis. In the further part of the report there will be a detailed description of the requirements. Having it allowed us to create the analysis. As the plan of the implementation is drawn, we are allowed to choose test strategy and approach and generate a test plan.

3.4.4 Test strategies description

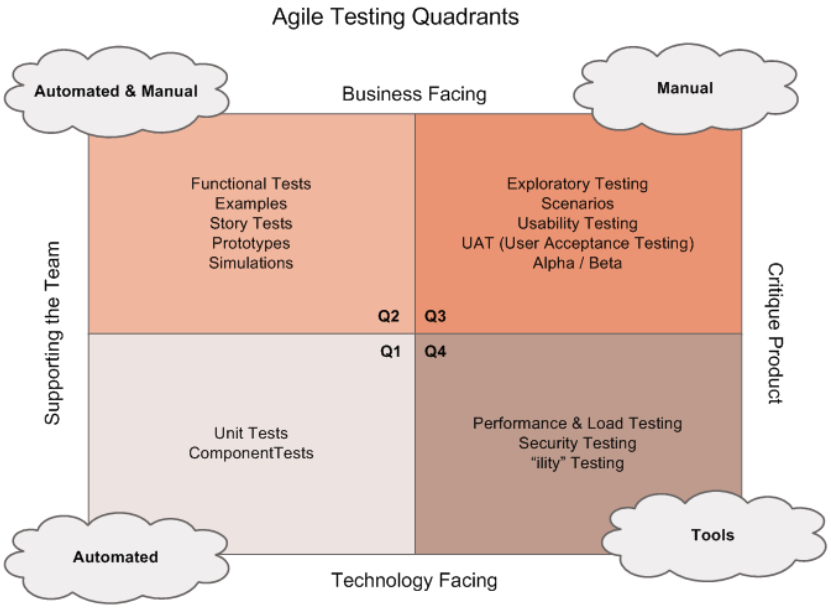
At the initial stage of a development cycle of a project it is essential to choose a testing strategy that would fit the most in our situation. There are three testing strategies that we could use. They are:

- V-model – consists of verification phases. Each step of the model: requirements analysis, system design, architecture design and a module design have its own verification step. The model is suitable for the scenarios when requirements are well defined and clearly documented, product definition is stable, and the project is short. However, if any changes happen in midway then the test documents together with requirement documents has to be updated which might create loop.

- Waterfall – in the model each phase must be completed before the next one can be started. One of the phases is testing. It is placed after implementation. During waterfall it is not recommended to go back to the previous phase. It is very simple and easy to use. The main disadvantage of the model is that there is no working software produced until the late during the life cycle.

- Agile – the model includes an incremental approach to testing. Tests need to be implemented early and often. That means they happens continuously as features are added. As prioritizing user stories, in an agile test strategy, tests need to be prioritized as well. Development team aims to write as many tests as possible during an iteration.

Depending on the stage of the development, tests from different quadrants might be chosen. Planning stage would aim to Q2 quadrant. At the early stages of the development would mainly use tests from Q1. As the work on the project continuous we will be taking advantage from tests from Q3 and Q4 quadrants.



3.4.5 Test Strategy selection

The work on the project is divided into several releases because of the time and resources limitations. Because the development methodology that was used for the scope of the project is agile, by combining waterfall or v-model testing strategies we would not benefit from them much. Despite the fact that all the requirements for the project are known, we expect that during the implementation some new would appear. Consequently, the project will have rapid characteristics and a lot of changes might happen. The two prescriptive models (waterfall and v-model) do not support such circumstances.

Waterfall model focuses on a one-way development. That means it is not recommended to go back to the implementation stage while writing tests. Our intention is to add vertical slices of the software - go back to certain components as we are adding new features and change/adjust – that is impossible with waterfall. V-mode would not let us control and eventually change the development process whenever we would like to, instead we would have to complete the complete planned cycle, both (verification and validation). Consequently, there is a risk of making errors at different development stages, mainly at the requirements interpretation. In the changing development environment, as the project is expected to be, it would be very expensive to repair failures. To the contrast to that, we would like to prevent the failures as soon as possible rather than repairing them. For that reason, it is essential to ensure the quality and reliability of a code at each release. That is why we decided to use the agile testing strategy.

The strategy allows to prioritize and choose the most essential components of the program that would be tested. Consequently, it is possible to set a done criterion for testing, which will help the team to trace a progress of a development. Agile strategy distinguishes four separate quadrants that aims to categorize different tests to their facing purposes. This becomes helpful while planning and executing.

3.4.6 Test plan Georgia Tech Library

1. Product and Overview

   Product is an ordinary software for library processes in this case for GTL. The software is allowing its users to browse, loan and return books as well as communicate to order books from other libraries through exposed endpoints.

2. Product History

   This product is in its initial phase, and has no previous record

3. Project scope

   The project scope is already known and well defined.

4. Project Objective
   Our test's main objective is to find defects and assure quality of the released product. We will also put a high focus on the requirements aspect. While outlining the requirements we will create acceptance criteria thanks to which the development team will know what is expected from the feature to do. Consequently, we will know that the end result meets the business expectations.

5. Features to be tested

   As we are prioritizing the tasks and implementing most crucial in given iteration as first, the same we will do with tests. It is not possible to test every single functionality. Moreover, we have limited time resource. Therefore, we would like to at most test the features, which defects can spoil the functionality of the whole system. In that way we will reduce a number of high severity defects that can occur. We decided to do testing at the same time as the implementation is written. By doing that we will make sure that as the new functionalities are going to be added, the previous version of system is still working without any complaint.

6. Features not to be tested

   Contrary to previous point, the features that are of low risk and without major impact on the software quality are going to remain untested.

7. Environmental requirements

   In order to test implemented software, we will use:

   - A physical machine with Java and Node.js installed
   - A blue ocean database available

8. Test approach

   The way tests are going to be processed is that for every desired functionality we are going to write test cases scenarios. Doing that will give a clear overview of under what circumstances was the method or component tested. Therefore, we will be able to determine whether it is worth creating more tests for a certain part of the software. Depending on type of tests, we are going to use different tools. Regarding dynamic testing, since the API is going to be written in Java, we will use Junit framework and Mockito library. While testing client application we were provided with external testing tool called SeqZap which is going to allows to script users actions and expect certain things to happen. Talking about static tests we will review each other code on daily basis, and making informal meeting where people talk about what have been done and the approach that was used for archiving a certain solution to a problem. Therefore, we are forced to outline some roles and responsibilities. Because of the project's nature we are limited in human resources. That is why all of us takes roles of test manager or test engineer. During our meetings we will change the roles. Consequently, some good ideas about testing or evaluating the code can come along. Because of the vertical slices approach, we knew we need to ensure that our new code does fail already existing components. One of the ways that Agile handles that kid of development is regression testing (we will test code that has been already tested and past, on every iteration) precisely automated regression testing**. In our project - Agile its success condition is the extent to which regression testing is automated.** Our development structure is more prone to have changes that affect our meta code and as the times go next iterations have passed, there is going to be more and more components that are going to fall under the regression test area. That is why we decided that in Q1 we are going to automate our regression tests and integration tests using IntelliJ IDEA combined with Junit. In Q2 of Agile we want to focus on story tests and simulations that reflect actual work of the librarians and users. To do that we took advantage of SeqZap tool. In the Q3 phase we will focus on manually playing around with different parts on the software by using implemented client. We will validate our software in Q4 using tools like Postman e.g. checkout the security.

   Risk driven development - We base our implementation on risks. Therefore, we would like to implement most complex features at the beginning of the project. Consequently, by using agile testing strategy, appropriate tests will be created parallelly to the features' implementation. The purpose of that is to ensure that while writing an implementation it is continuously verified.

8.1. Entry Criteria

The software must meet the criteria below before the product can start system test. Specifically enumerate any project-specific departures from the Generic Criteria. This list must be negotiated with and agreed upon by the project leader:

1. All functionality related with borrowing and returning the book must work.

2. All unit tests, integration run without error.

3. The code is frozen and contains complete functionality.

4. All code compiles and builds on the appropriate platforms.

8.2. Exit Criteria

The software must meet the criteria below before the product can exit from system test. Specifically enumerate any project-specific departures from the Generic Criteria. This list must be negotiated with and agreed upon by the project leader:

1. Coverage for tested functionality is above 70%.

2. All automated tests execute

3. System manual testing execute

   3a) Successful execution loaning a book.

   3b) Successful execution of returning a book.

4. Any failing test must be discussed and fixed (in e special case we will accept the fail).

5. Code is frozen.

9. Test Deliverables

   • Automated tests in SeqZap and IntelliJ IDEA

   • Test procedure

   • Test logs

   • Test Coverage measurement

3.5 Development stage

3.5.1 Requirements definition

Having defined the features that will be in the scope of our project's release, we were able to create a requirements definition. We had to specify the functional requirements at most detailed way as possible. It is because they need to be understood correctly by all member of the development team. Otherwise, if the requirement would be misunderstood, and the implementation would be based on such understanding, then the feature is not providing what a client is expecting from us. Furthermore, it would be very costly to fix the feature, because at that time its problem lies on the side of requirements. Consequently, to refactor it, in order to meet expectations, it would require remaking both design and implementation stages. For that reason, testing at the very beginning of the project stage is very essential and important. From the functional requirements that we have gathered, we have created user stories. Inside of which we estimated number of hours that need to be spent on the implementation, importance of the task as well as acceptance criteria. The importance is measured from one to four. The higher a number is, the more important a feature is. The criteria were made so that both sides, meaning development team and customer would agree that if they are met, then the requirement is complete. Even though we have our requirements set clearly, we might expect that some new additional ones will appear. However, the ones that we have outlined are the following.

1. Loan a book

ID:1	**TITLE: LOAN A BOOK**	
AS	Librarian	
I WANT	to loan a book to a member	
SO THAT	they have borrowed for certain period	
ACCEPTANCE CRITERIA	*Acceptance Criterion 1:* • The book state is changed for lend and it cannot be borrowed by any other person *Acceptance Criterion 2:* • The book is assigned to the member's borrows and displays at his profile *Acceptance Criterion 3:* • Librarian can see a new borrowing record with all relevant information about it. *Acceptance Criterion 4:* • Borrowing time has started	
	ESTIMATED SP: 12	IMPORTANCE: 4

The user of the requirement is a librarian. Librarians should be able to loan a book for user at any time. Having done that, the state of the book is changed and the time of the borrow has been saved as well. Consequently, no other person would be able to borrow it until it is returned. Moreover, the book is assigned to the borrower's borrow record, together with his card number. In result it is possible to see, who is currently having a certain book. That means, a librarian while navigating to a page where all current borrows are displayed, will see a new record about the borrow that they have made. We estimated the user story to be implemented in twelve hours and it has the highest possible importance score.

2. Borrow a book

ID:2	TITLE: BORROW A BOOK	
AS	Member	
I WANT	to be able to borrow a book	
SO THAT	I can acquire the knowledge they hold	
ACCEPTANCE CRITERIA	*Acceptance Criterion 1:* • Member can successfully borrow a book, the book is saved in his history and no other user can borrow it, unless there are still copies. *Acceptance Criterion 2:* • Book availability is changed *Acceptance Criterion 3:* • Librarian can see a new borrowing record with all relevant information about it. *Acceptance Criterion 4:* • Borrowing time has started	
	ESTIMATED SP: 8	IMPORTANCE: 3

This user story is very similar to the previous one. The only thing that is different is a person who makes action. Previously it was a librarian who was loaning books for student. This time it is student themselves who makes actions. All of acceptance criteria are the same. Nonetheless, the estimated time of completing the feature is smaller. It is because we would probably take advantage from already existing implementation from the first user story.

3. Displaying loans

ID:3	TITLE: LOANS OVERVIEW	
AS	Librarian	
I WANT	To be able to see currently borrowed books and history of borrowings	
SO THAT	I can keep track with what I have borrowed	
ACCEPTANCE CRITERIA	*Acceptance Criterion 1:* • Member can see all previous borrowings *Acceptance Criterion 2:* • By borrowing a new book, it is added to the history *Acceptance Criterion 3:* • Status of borrowing is also available	
	ESTIMATED SP: 8	IMPORTANCE: 3

It should be also possible for librarians to see borrowing history of all users' currently borrowed books as well as currently borrowed books. There could be a button, that opens a page where a librarian can see a table with such information. The user story is considered as functional when all previous borrowings from history on the first table, and currently unavailable books on the second table are displayed. Moreover, a book should be immediately be displayed on the table that traces books being borrowed, whenever a new borrow is made. To distinguish currently borrowed books from the ones that have been returned, a special column determining status of the loan should be also available. It is quite important feature for the library.

Thanks to it, librarians have possibility of tracing how many books are not available on loan as well as, who are the people that have books borrowed. We decided to estimate the implementation of that feature for 8 hours.

4. The view of a book catalog

ID:4	**TITLE: CATALOG**	
AS	Librarian	
I WANT	Have insight into the catalog of books and search for books	
SO THAT	I can see what books are available and easily find them	
ACCEPTANCE CRITERIA	*Acceptance Criterion 1:* • The librarian can see all the books together with all its information • The member can see only limited amount of information and all the books *Acceptance Criterion 2:* • Applied filters should have an impact on result of searching *Acceptance Criterion 3:* • Displayed catalog should be reliable, which means that displayed number of copies need to be constantly up to date.	
	ESTIMATED SP: 8	IMPORTANCE: 4

The next feature that is going to be implemented is a view of books catalog. Providing the feature in a release product is essential. Thanks to that, it is possible for librarians to distinguish different copies of the same book as well as search desired by student's books. The searching can be done by book's author or title. The feature can be considered as done when the list of all books with limited number of information is shown to a system user. Moreover, it should be also possible to apply filters that would limit number of displayed entries. Last but not the least criterium is that the number of currently available copies for a searched book is reliable. That means there should not be a case when a system shows that there are books that can be borrowed but in fact there is any. The feature is estimated to be implemented in 8 hours.

5. Returning a book

ID:5	TITLE: RETURN A BOOK	
AS	Librarian/Member	
I WANT	To state the book of a member as returned	
SO THAT	It can be borrowed by other people	
ACCEPTANCE CRITERIA	*Acceptance Criterion 1:* • The book's status is changed and can be borrowed by other members *Acceptance Criterion 2:* • Availability is altered *Acceptance Criterion 3:* • Member's active borrowing is terminated, return date was saved *Acceptance Criterion 4:* • System checks if the book was returned withing right time and alternatively calculates a fee that need to be paid	
	ESTIMATED SP: 8	IMPORTANCE: 4

While borrowing a book is going to be provided in the system implementation, then the logical thing is that it should also provide a functionality of returning books. The action's user can be either a librarian or a student. Having returned a book, its status must be changed, which means that its number of availability copies was incremented by the one that has been returned. Consequently, other users of the system are allowed to borrow it. Moreover, user's borrowing record was terminated, status of the borrowing is set to finished and a returned date has been saved. At the same time, the system should check if the user returned the book within right period. If not, it should calculate a fee to be paid.

6. Distinguish materials that cannot be lent

ID:6	TITLE: DISTINGUISH MATERIALS THAT CAN NOT BE LENT	
AS	Librarian	
I WANT	To see if a certain material can be lent	
SO THAT	Well treasured materials will be safe	
ACCEPTANCE CRITERIA	*Acceptance Criterion 1:* • There is no opportunity for librarians or members to lean/borrow the material *Acceptance Criterion 2:* • Librarians are able to navigate to a page where a list of not for lent materials are displayed	
	ESTIMATED SP: 4	IMPORTANCE: 2

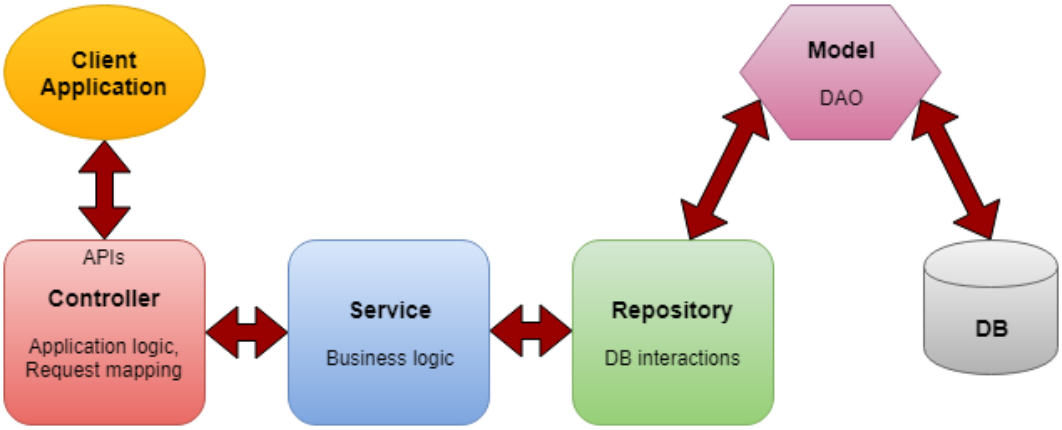
As it was described in the case description, there are some materials that should not be available for borrowing at any time. They are maps, reference-books, and rare books. Nonetheless, the system should store them in a memory for the sake of traceability. The user story of distinguishing materials that cannot be lent is considered to be done when there is no opportunity in the system for librarians to lend these materials. However, they can see what the materials and information about them are. The task is not much important. It has only an informative characteristic. That is why it has a low importance point. The estimated time for implementing the feature is four hours.

Mentioning non-functional requirements that we would like to aim during the project's implementation, they are:

- **Quality**
  We will be providing quality by implementing static and dynamic tests from the very beginning of the project lifetime. Moreover, we have created testing strategy and test each implementation, having implemented it. To strengthen the quality, we will also make our best to document the source code and we will also try to catch exceptions as much detailed level as possible. Then whenever, a defect occurs, it will be easy to trace it and repair the implementation logic as fast as possible.
- **Security**
  We will be using a Spring framework while creating an API application. The framework has a build-in library for security, which allows setting up restrictions to different endpoints of the application. One of its features is also protection against attacks like session fixation, clickjacking, or cross site request forgery. Despite, that we will also protect our database by creating different level user logins. Some of them would be provided only with limited permissions so that they can process only necessary queries to complete tasks they should.
- **Testability**
  The structure of the server implementation will have testable structure. We will be using dependency injection. Consequently, it will be possible to mock different components of the implementation in order to test only the desired piece of code.
- **Efficiency**
  If we are going to be dealing with a huge amount of data while processing users' queries, we will handle it for example by splitting a result set into pages and returning only a fraction of the whole data back to user. Then a user can process a new request to the server, pass a parameter that would filter the whole database and choose the appropriate subset. Consequently, the software would work faster. Moreover, we will improve efficiency by adding clustered and non-clustered indexes in a database on an appropriate column. In result, the database will be able to faster process queries.
- **Reliability**
  We will ensure reliability by testing the application. On the other hand, we will be using transactions. Thanks to them, while processing user's request, if there will be more than one call made to a database and for example second one fails, then the previous changes will be discarded and rollbacked. Consequently, the state of the data source would be unalterable, even though some of the database queries succeed.

3.5.2 Design

As a development team, we have decided to build and an API that will be accessed from outside by a client application. Therefore, we are going to deliver both things to the customer. We focused on designing a structure of an API that would be as most testable as possible. The API structure is based on a repository pattern. It consists of three layers that handle different responsibilities. They have been already described while presenting objectives for the project. Nonetheless the flow of events in the design is the following.



To make the structure of the program testable we used dependency injection. It means that a layer uses injected interfaces in order to process the users request further. For example, each service class is implementing an appropriate interface. The injected interface is instantiated in the controller's constructor. Consequently, the call is not handled directly by the service class implementation but a through its interface as illustrated below.

```
public class BookCatalogController {

    private IBookCatalogService iBookCatalogService;

    private IAvailableBooksService iAvailableBooksService;

    @Autowired
    public void setIAvailableBooksService(IAvailableBooksService iAvailableBooksService) {
        this.iAvailableBooksService = iAvailableBooksService;
    }

    @Autowired
    public void setIBookCatalogService(IBookCatalogService iBookCatalogService) {
        this.iBookCatalogService = iBookCatalogService;
    }

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public ResponseEntity<?> getBooksCatalog() {
        return new ResponseEntity<>(iBookCatalogService.getBooksCatalog(), new HttpHeaders(), HttpStatus.FOUND);
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public ResponseEntity<?> getBookCatalog(@PathVariable int id) {
        return new ResponseEntity<>(iBookCatalogService.getBookCatalog(id), new HttpHeaders(), HttpStatus.FOUND);
    }
```

The IBookCatalogService look the following.

```
public interface IBookCatalogService {
    List<BookCatalogReturn> getBooksCatalog();
    BookCatalogReturn getBookCatalog(int id);
    BookCatalogReturn addBookCatalog(BookCatalogCreation bookCatalogEntity);
    BookCatalogReturn updateBookCatalog(BookCatalogUpdate bookCatalogEntity);
    boolean removeBookCatalog(int id);
}
```

And the implementation class that is being injected during the start-up of the program is:

```java
@Service
public class BookCatalogService implements IBookCatalogService {

    private IBookCatalogRepositoryCustom iBookCatalogRepositoryCustom;

    private IBookService iBookService;

    private ModelMapper modelMapper;

    @Autowired
    public void setIBookService(IBookService iBookService) { this.iBookService = iBookService; }

    @Autowired
    public void setModelMapper(ModelMapper modelMapper) { this.modelMapper = modelMapper; }

    @Autowired
    public void setIBookCatalogRepositoryCustom(IBookCatalogRepositoryCustom iBookCatalogRepositoryCustom) {
        this.iBookCatalogRepositoryCustom = iBookCatalogRepositoryCustom;
    }

    @Override
    public List<BookCatalogReturn> getBooksCatalog() {
        return iBookCatalogRepositoryCustom.getBooksCatalog();
    }
```

The reason for using dependency injection is to allow mocking some of the system components while writing tests. That means that during testing implementation of a specific service method it will be possible to mock its methods calls to the database. Consequently, it will ensure that the thing that is tested is a service itself rather than other parts of the system.

Another design approach, that supports testing in our program is creating a separate database and connect to it only when tests are ran. While testing repositories calls and verifying that the methods work correctly, we have to manipulate records manipulate a database. To do that we have created another instance running against which we will be running test queries. Consequently, we will not spoil data from the original database. Moreover, while the product will be released and used the tests could be still executed. These design constraints were related to the API implementation. Mentioning the client application its structure will be based on services that sends request and receive response from a server, process the data and put them in an appropriate manner on views. By using SeqZap tool, we can script users' actions and assert appropriate results to be displayed.

3.5.3 Implementation

Since we based our implementation on creating a feature that are most risky at the beginning, then before implementing anything, we had to find the riskiest feature. To do that we have made a risk analysis.

Risk product	Probability [1-3]	Impact [1-3]	Risk score	Reflecting requirement [ID]
The user is not able to borrow/loan a book, even though it is available to be borrowed/leant	1	3	3	[1,2]
The borrowed book's state is not change. Consequently, it is still marked as available for other users	2	3	6	[1,2]
Having borrowed a book, it was not assigned to the right user or not assigned at all	2	2	4	[1,2,3]
Librarian is not able to see the newly created record of a borrowed book	1	2	2	[1]
Having borrowed a book, a new record was added to borrowing however the start time was not set	2	3	6	[1,2]
The status of the borrowing is not displayed at all or it is not correct	2	2	4	[3]

The history of borrowing is not being displayed	1	3	3	[3]
There are missing relevant information about a book in a book catalog overview	1	2	2	[4]
Employee sees information about books that should not have been	1	1	1	[4]
Filters on the catalog overview does not work	2	1	2	[4]
Catalog wrongly displays number of available copies	2	2	4	[4]
The book availability has not been changed while returning a book and it cannot be borrowed by anyone	2	3	6	[5]
The borrowing has been terminated but the returned date is not set	2	2	4	[5]
The borrowing has not been terminated	1	3	3	[5]
Calculation of a fee does not work	2	2	4	[5]
Librarians are able to borrow a material that they should not be able to	1	3	3	[6]
List of not for lent materials is not working	1	1	1	[6]

The table above contains risks related with an implantation of planned features. By accumulating risk score for each of the features we can prioritize them. From most risky to the least, as a result we get:

Requirement name	Points
Loan a book	21
Borrow a book	19
Return a book	17
Displaying loans	11
The view of a book catalogue	9
Distinguish materials that cannot be lent	4

Having the results, we have an overview what should be an order of implementing the features in the software. Based on that the first implementation is concerning loaning a book. Having made the implementation, we were about to test it at the different levels. To do that, it is required to create a test cases scenario. They will help us to trace the progress and help us implanting some well-defined tests. Because of time limitations we wanted to test only most risky features of the system. The three last features, which are displaying loans, the view of a book catalogue and distinguish materials that cannot be lent are not going to be included in the scope of automated tests. Nonetheless, we will system test these features from the client application.

3.5.3.1 Repository tests BookBorrow Integration Tests

The method that handles adding a borrowing to a database is called createBookBorrow. It takes two parameters of type BookBorrowEntity and int. The first one is an entity object instance and the second one is an id of a created book return record. BookBorrowEntity object has 5 attributes, from which only 2 should be provided. They are bookCatalogId, that is a reference to a book copy from a database and a ssn which is a number that identifies user borrowing a book.

	Test Scenario	Values	Expected	Software reaction
Repository tests				
1.	Object of type BookBorrowEntity is passed as a null value	BookBorrowEntity = blank Id = valid	Should throw NullPointerException	Throws NullPointerException
2.	Object of type BookBorrowEntity is missing user's ssn value	BookBorrowEntity = invalid (missing ssn) Id = valid	Should throw CreationException	Throws CreationException
3.	Object of type BookBorrowEntity has invalid user's ssn	BookBorrowEntity = invalid (invalid ssn) Id = valid	Should throw CreationException	Throws CreationException
4.	Object of type BookBorrowEntity is missing book's copy id value	BookBorrowEntity = invalid (missing bookCatalogId) Id = valid	Should throw CreationException	Throws CreationException
5.	Object of type BookBorrowEntity has invalid book's copy id value	BookBorrowEntity = invalid (invalid bookCatalogId) Id = valid	Should throw CreationException	Throws CreationException
6.	All parameters are valid	BookBorrowEntity = valid Id = valid	Should add to database and return BookBorrowReturn object	Returns BookBorrowReturn object and adds record to database

3.5.3.2 Service tests BookBorrow Unit Tests

The method that is responsible for handling borrowing a book in service class is named borrowBook. It takes one parameter of type BookBorrowCreation. In order to correctly process the method, the object's bookCatalogId attribute must be filled. The attribute refers to the id of a book's copy.

	Test Scenarios	Values	Expected	Software reaction
Service tests				
1.	Invalid value of the bookCatalogId attribute in a BookBorrowCreation object	BookBorrowCreation = invalid (invalid BookCatalogId)	Should throw NotFoundException	Throws NotFoundException
2.	There are no book copies available to be borrowed	BookBorrowCreation = valid Mock – IbookService.findBook returns BookBorrowReturn object with 0 copies available	Should throw NotFoundException	Throws NotFoundException
3.	The book is not available to be borrowed	BookBorrowCreation = valid Mock – IavailableBookService.findAvailableByCatalogId returns null	Should throw NotFoundException	Throws NotFoundException
4.	There was an error during the creation of BookReturn record	BookBorrowCreation = valid Mock BookReturnService.createBookReturn returns CreationException	The transaction is rolled back and CreationException is thrown	CreationException is thrown and no data is inserted into the database
5.	Valid data positive scenario	BookBorrowCreation = valid	Should returns BookBorrowReturnView	Returns BookBorrowReturnView

3.5.3.3 Controller tests BookBorrow Integration Tests

In the BookBorrowController there is one endpoint implemented which allows to borrow a book. It requires the BookBorrowCreation object passed as JSON. It holds the information about the book catalogue id of the book we want to borrow. There are several scenarios the software must be secured from.

	Test Scenario	Values	Expected	Software reaction
Controller tests				
1.	The bookCatalogId is empty	bookCatalogId = empty Perquisites Token must be passed in the header	Should return in the response: "ID of the catalog must be set"	Returns: "ID of the catalog must be set" as the response
2.	The input is not numeric	bookCatalogId = abc Perquisites Token must be passed in the header	Should return bad request	Returns bad request
3.	The input is not valid	bookCatalogId = -1 Perquisites Token must be passed in the header	Should return in the response: "There was an error while finding a book catalog with ID: -1"	Returns "There was an error while finding a book catalog with ID: -1" as a response
4.	One tries to borrow the same book multiple times	bookCatalogId = 18 Perquisites Token must be passed in the header	Should return in the response: "The book was not found or is not available"	Returns "The book was not found or is not available" as a response
5.	Valid borrowing	bookCatalogId = 18 Perquisites Token must be passed in the header The book must not be borrowed	Should return the information about the borrowed book	Returns all information about borrowed book

3.5.3.4 Repository tests BookReturn Integration Tests

Next crucial method in the software allows to return a book. It is called returnBookAndChangeStatus. It takes one parameter id. It identifies the BookReturning record from the database. Going further after the specific BookReturn record is found the software updates the returned_date field to the current date. In the meantime, a method calculatePayment which takes two date parameters calculates the difference between them in days. If the result is bigger than 0 then it is multiplied by a set value of charge for a single day of delay and returned. Therefore, together with changing the status the update includes returned_date and payment field.

	Test Scenario	Values	Expected	Software reaction
Repository Tests				
1.	The id is valid	Id = valid	Should return true and insert into database	Returns true and inserts
2.	The id is invalid	Id = invalid (-1)	Should throw UnknownException	Throws UnknownException
3.	The payment is calculated right	Id = valid Perquisite A BookReturn record with earlier date by 10 days than current is in the database and we know its id	Should return true and the payment should be equal to 7.5	Returns true and the payment is equal to 7.5

3.5.3.5 Service tests BookReturn Unit Tests

In the BookReturn contains returnBook method which must be tested which was implied by the risk analysis. It takes in one parameter which is a BookReturnCreation object. The possible scenarios which can occur are taken into consideration and based on them following test scenarios are produced:

	Test Scenario	Values	Expected	Software reaction
Service tests				
1.	Invalid bookCatalogId while returning book	BookReturnCreation = invalid bookCatalogId Mock IbookCatalogService. GetBookCatalog returns NotFoundException	Should throw NotFoundException	Throws NotFoundException
2.	Valid data but unknown error during the returning	BookReturnCreation = valid Mock IBookReturnRepository. returnBookAndChangeStatus returns false	Should throw UpdateException	Throws UpdateException
3.	Valid data positive scenario	BookReturnCreation = valid	Should return true	Returns true

3.5.3.6 Controller Tests BookReturn Integration Tests

The BookReturn controller contains two endpoints and one of them provides the functionality to return the book. The risk analysis conducts to test as a one of the highest priorities. Therefore, several test cases. Although the class and endpoint are different, the functionality it provides and the data it awaits is similar to the BookBorrow controller. Therefore, the tests scenarios are nearly identical.

	Test Scenario	Values	Expected	Software reaction
Controller tests				
1.	The bookCatalogId is empty	bookCatalogId = empty Perquisites Token must be passed in the header	Should return in the response: "ID of the catalog must be set"	Returns: "ID of the catalog must be set" as the response
2.	The input is not numeric	bookCatalogId = abc Perquisites Token must be passed in the header	Should return bad request	Returns bad request
3.	The input is not valid	bookCatalogId = -1 Perquisites Token must be passed in the header	Should return in the response: "There was an error while finding a book catalog with ID: -1"	Returns "There was an error while finding a book catalog with ID: -1" as a response
4.	One tries to borrow the same book multiple times	bookCatalogId = 18 Perquisites Token must be passed in the header	Should return in the response: "Book is not borrowed"	Returns "Book is not borrowed" as a response
5.	Valid borrowing	bookCatalogId = 18 Perquisites Token must be passed in the header The book must not be borrowed	Should return true	Returns true

Although in our case the development cycle was short, the final number of tests implemented is 76. In the real-world situation and a full development period there can be much more tests. Consequently, it is impossible to run all of them manually. Therefore, there are many tools available on the market which can start and evaluate all the tests in our software, however thanks to the choice of IntelliJ IDEA none of the external software must be used. It provides the functionality to run all the tests implemented just by two clicks. All the results are visible afterwards in the console. Although, the tests are located in different directories or IDE is capable of finding them and executing. The only requirement is that they must be under the "test" folder created while the project is initialized by the IDE.

3.5.3.7 User Interface Tests System Tests

Another important factor is testing the user interface against different scenarios to be sure that the user is informed about the actions the system is undertaking. Furthermore, we can test the whole system including database just by testing the client. Therefore, we came up with several test cases regarding the flow of borrowing and returning the book.

	Test Scenario	Values	Expected	Software reaction
User Interface tests				
1.	User borrows one book	Book catalog id valid	Should display info about the book at the bottom	Displays info about the book at the bottom
2.	User returns one book	Book catalog id valid	Should delete info about one book from the bottom list	Deletes info about one book from the bottom list
3.	User borrows two books	Book catalog ids valid	Should display info about two books at the bottom of the page	Displays info about two books at the bottom of the page
4.	User returns two books	Book catalog ids valid	Should delete info about two books from the bottom list	Deletes info about two books from the bottom list
5.	User borrows book with incorrect book catalog id	Book catalog id = invalid	Should display message "Incorrect book catalog id"	Displays message "Incorrect book catalog id"
6.	User returns book with incorrect book catalog id	Book catalog id = invalid	Should display message "Incorrect book catalog id"	Displays message "Incorrect book catalog id"

Manual testing in this case which includes only basic functionalities would be expensive. Therefore, we decided to use a SeqZap tool. It allows to automate every step such as clicks or inputs and performs all the tests for us. Thanks to the tool we created automated system tests.

3.5.4 Exit criteria evaluation

Test coverage

Before completion stage we had to ship the product to the customer. That requires us to meet the exit criteria. One of the conditions is the test coverage for the components we are developing in this iteration. As mentioned previously we decided to test using risk-based strategy. After implementing and executing all of the testing we ended-up with following results. Within the iteration risk-based components (book borrow and book return) we managed to cover 144 lines out of 155 lines and that gives us 93,5 % test coverage. This is more than required therefore we accomplished what we wanted (more than 70%).

Element	Class, %	Method, %	Line, %
BookBorrowRepository	100% (1/1)	100% (4/4)	86% (20/23)
BookBorrowService	100% (1/1)	100% (17/17)	100% (45/45)
BookBorrowController	100% (1/1)	100% (2/2)	100% (4/4)
BookReturnRepository	100% (1/1)	100% (7/7)	84% (32/38)
BookReturnService	100% (1/1)	100% (8/8)	100% (38/38)
BookReturnController	100% (1/1)	75% (3/4)	85% (6/7)

For the entire project on the other hand (so also including functionality that was out of the test scope based on the risk analysis) we manged to cover 1076 lines from 2291 lines that gives 46,9%

Element	Class, %	Method, %	Line, %
Controllers	100% (12/12)	46% (27/58)	64% (61/95)
Exceptions	71% (5/7)	71% (5/7)	71% (10/14)
Handlers	94% (17/18)	75% (41/54)	83% (161/193)
Models	46% (30/65)	39% (245/618)	31% (402/1261)
Repositories	70% (17/24)	68% (75/109)	55% (273/488)
Services	93% (14/15)	65% (63/96)	70% (168/237)
MainApplicationClass	100% (1/1)	0% (0/1)	33% (1/3)

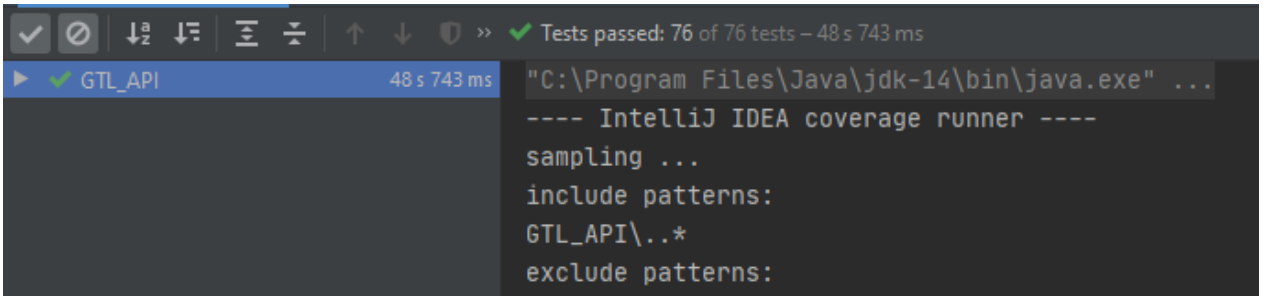
Element	Class, %	Method, %	Line, %
GTL_API	67% (96/142)	48% (456/943)	46% (1076/2291)

System manual testing execute

At first, we were encountering platform related problems when running manual tests that were validating the book borrowing and returning process. But fortunately, we manged to solve problem with environmental variables.

Any failing test must be discussed and resolved

During test we have faced problems with an authentication token. It was necessary for integration tests to first provide a token in a header and then send a request. Another issuer that we have come across was while testing services methods of borrowing and returning books. It was necessary to mock a logged user so that we get his username, which is crucial for processing the operation of borrowing and returning books.



Code is frozen

In our case we do not freeze the code completely as in future we will resume working on it and meet other requirements such as statistics and external functionality of the program. Instead as the team is small and communication direct, we decided to stop everyone from changing the software for the delivery time of the product.

Tests traceability

Whenever a customer issues a problem with a software, it is possible for us to run tests and determine if they pass. In case when some of them fails, by implying naming convention and catching/throwing detailed exceptions at different levels of structure, it would be easy to find what is the root of the problem and which component of the software is failing.

3.6 Completion stage

Having implemented and tested the desired features, the software is ready to be delivered to a customer. When a customer receives the product, it is important so that we would help them get familiar with it and train how to use it. If it were a real-life project with existing customer, we would invite them to run factory testing, which are tests performed during the development process. Afterwards, customer by trying the program's functionalities, runs acceptance tests. They validate if the software does what was expected and agreed. Customer can accept or decline what was created. Depending on a case, the product owner can either accept what was delivered or instruct people who were responsible for creating the program to fix the bugs in agreed time range.

3.7 Operational stage

As the customers familiarize themselves with the software, they will require small enhancements as well as well as spot minor bugs. In case of our project this was not part of the requirements therefore not guarantee was agreed upon the project. Nevertheless, it is more likely that in real-life scenario such feature would be requested and be part of the project. How we would handle it would be to assign part of our team to take care of enhancements and different one to take care of bugs.

3.8 Post-project review

After couple of months from delivering the product to the customer, we would review the whole project progress. It is essential to check if the agreed objectives, business, and deliverables constraints have been met. This stage is done so that we could reflect on what went good and bad. Consequently, we benefit from that because in the next project we could put a bigger focus on things that went bad and improve that.

3.9 Conclusions

Just like in any other project we started by designing the system development part but this time we put a lot of focus into testing with respect to the time management/resources. Having limited experience in static testing processes we tried to simultaneously learn and develop our strategy. The testing form we have finished our project with is a result of discussion and exploratory analysis of 4 team members. We decided on what to test (risk based implementation) in the project plan and followed that thought the project. In our opinion it is a suitable way to carry out testing activities based on our skills and knowledge. Thanks to the later stages of the testing course we are able to create detailed plans for dynamic testing techniques in our project. This came in handy when tracking and evaluating our test`s results – good traceability.

Though we are aware of the alternative paths(e.g. requirements based testing) we could have picked up, during this project we inspired our work on the lectures we have been attending. There we have been told that part of the testing is about making assumptions. In organisations the more project you carry out the easier it is to get closer to right solutions with the assumptions.

We think that this statement is crucial and true both in our case. In future we will be able to benefit from this project and achieve better results using the same amount of time. When it comes to testing techniques that will directly increase the amount of features we will implement and test. We have learnt how to carry out entire testing process for our software and that will be beneficial for us in our later career paths as well as future projects.

Future work of this project issues the remaining features such as statistics and incorporation of external APIs into our software. The team is happy with the results and wants to continue the work. We hope to finish with all of the requirements and use this project for future job search/portfolio.

3.10 References

1. https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm (703)
2. https://www.toolsqa.com/software-testing/waterfall-model/
3. http://tryqa.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/
4. https://www.guru99.com/non-functional-testing.html
5. http://softwaretestingfundamentals.com/white-box-testing/
6. https://asq.org/quality-resources/software-quality
7. https://www.softwaretestinghelp.com/test-plan-sample-softwaretesting-and-quality-assurance-templates/
8. https://www.softwaretestinghelp.com/writing-test-strategy-document-template/
9. https://eternalsunshineoftheismind.wordpress.com/2013/02/22/v-model-advantages-disadvantages-usage/
10. https://en.wikipedia.org/wiki/Black-box_testing
11. https://www.sealights.io/agile-testing/understanding-agile-testing-methodology-and-4-agile-testing-methods/
12. https://en.wikipedia.org/wiki/Project_plan
13. http://moodle.autolab.uni-pannon.hu/Mecha_tananyag/szoftverfejlesztesi_folyamatok_angol/ch12.html
14. https://searchcompliance.techtarget.com/definition/risk-management

15. https://www.pm4dev.com/resources/free-e-books/3-the-project-management-organizational-structures/file.html
16. https://en.wikipedia.org/wiki/White-box_testing
17. https://www.guru99.com/software-testing-seven-principles.html
18. https://www.softwaretestinghelp.com/test-plan-sample-softwaretesting-and-quality-assurance-templates/
19. https://en.wikipedia.org/wiki/Software_testing
20. https://strongqa.com/qa-portal/testing-docs-templates/test-strategy
21. https://stackoverflow.com/questions/1134319/difference-between-a-user-and-a-login-in-sql-server
22. https://www.concurrency.com/blog/january-2017/creating-logins-and-users-in-sql-server