UCN psu0219

1st Semester Project

Group 13

DATABASES & TEST REPORT

29th of May 2019

Andreas Richardsen

Zahro-Madalina Khaji

Contents

Database

Introduction

In this section of the report, we will discuss the Georgia Tech Library (GTL) case while focusing on database design and implementation. We analyse the requirements of the case, plan and document the process of building a suitable database solution. Throughout, we utilise the knowledge and skills gained from the Database Systems for Developers course. Furthermore, we present and explain the various choices and decisions we made. In the end, we will conclude with our final thoughts on the case and our implemented database solution.

Requirements

The GTL case gives us an overview of the library and its employees. The librarians describe their tasks and the ways in which the business rules of the library are enforced. They also mention some of the functionalities that they would like to obtain from the implementation of a software solution.

After carefully reading the case, we identified the following requirements:

1) Library wants to track the number of book copies that are currently on loan and those that are not on loan.
2) Library wants a list of all the books and their descriptions, which can be filtered by book author, title and subject area.
3) Regular library members can check out books for 21 days, can loan out a maximum of five books at a time and have a one-week grace period for returning borrowed books, before a notice is sent to them.
4) Librarians require a member's SSN, campus and home address, and phone numbers to register them into the system.
5) Each member gets a numbered library card with their photo on it, which is valid for four years after its issue, a month before the card expires, a notice is sent.
6) Professors can check out books for three months and have a two-week grace period.
7) Library wants a list of books that cannot be lent such as reference books, rare books and maps.
8) Library want a list of books that they are interested in acquiring but cannot acquire.
9) Library uses ISBN to uniquely identify books.
10) Library cooperates with other libraries and is interested in exposing statistics about them.

Features

From analysing the requirements, we selected a list of features for our databases solution. However, because of time and budget constraints, we have a limited amount of resources available to us. Therefore, we split these features into essential and non-essential ones.

Our list of essential features contains:

- The book loaning
- A view of all the book copies that are being currently loaned
- A view of all the books that are overdue
- A view of all the books that cannot be loaned according to the library's business rules
- A view of all the books that match a search criterion such as the book title

Our list of features that we will omit:

- A view of all the books that they are interested in acquiring
- The sending of notifications
- Exposing statistics about other libraries

The scope of this project is now determined and a plan for implementing these features is needed next. We start with choosing the number one priority, which for us is the book loaning feature. This is a core functionality that every library needs to have. GTL also specifies numerous business rules regarding this feature. This highlights its importance but also the complexity of its design and implementation. Therefore, the book loaning is what we will centre our solution around. For the rest of the essential features, we prioritise them in order that they are listed above.

Moving forward, we detail the process of designing and implementing our database solution.

Conceptual model design

The conceptual design of the database requires us to have an Entity-Relationship (ER) model and an Enhanced-Entity-Relationship (EER) model. These models are represented using diagrams. We use the ER and EER diagrams to map them into a database schema.

ER Diagram

The ER high-level conceptual model is used to model the important entities and relationships of our database solution. In this ER model we have the following four entities: Person, Card, Book and Map, and two relationships called Use and To_Borrow.

Entities

The **Person** entity models the users of the system which could be library staff or regular members such as students or professors. The key attribute of this entity is SSN attribute which is unique for every Person. The simple attributes are FName which stands for the Person's first name, LName which stands for the Person's last name and Type which stands for the type of user the Person could be. For the type attribute we have decided on the following values: Librarian, Student and Professor. The Person entity has a multiple valued attribute called Phone_No. In the requirements it was specified that the library would like to store multiple phone numbers associated to a Person. The composite attribute called Address has the sub-attributes Home and Campus. It was also noted that the library would like to store a Person's home address and campus address.

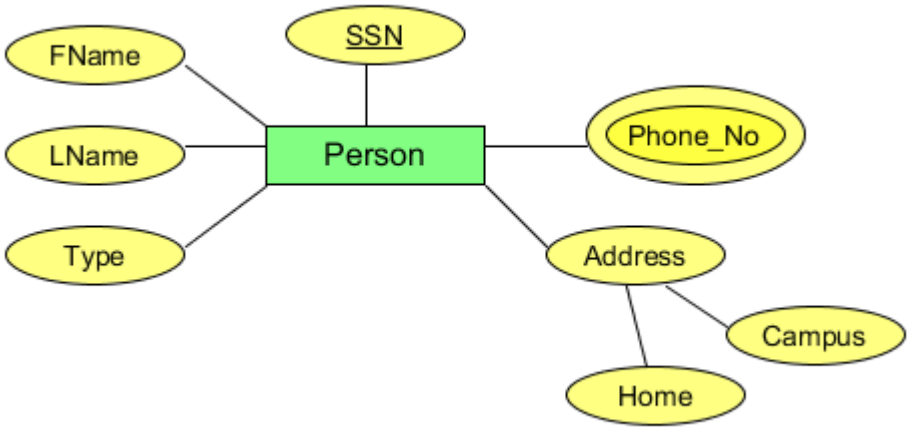

Figure 1: The Person entity in the ER diagram

The **Card** entity is used to model the card issued when a Person registers in the library system. The library assigns each card a unique card number. Card_No is the key attribute for the entity. We added the simple Card_Issue_Date attribute as we want to make sure that the system does not loan books (feature) to instances of the Person entity that have an expired card. Per the library's stated business rules, a card expires four years after its issue date.

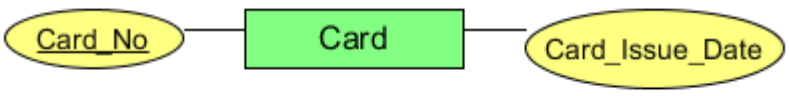

Figure 2: The Card entity in the ER diagram

The **Book** entity is a central entity for our model. The key attribute is the ISBN of a book, which is what the GTL library uses to uniquely identity its books. The simple attributes of the Book entity are Title, Author, Description, Publisher, Year_Publishing and Type. Except for the Type attribute, all of them are self-explanatory. The Type stands for normal, reference or rare book. The reasoning for choosing these three types will be explained as we continue a bit further down.

Book has two multiple-valued attributes called Subjects and Copies. We assume that each book would fall into one or more subject areas therefore we want to store multiple values for them. The Copies attribute is an interesting one. The library usually has multiple copies of a book at its disposal. For the loaning feature, we realised that we needed to keep track of the individual copies of books that can be loaned. To identify a copy of a book, the ISBN could not be used, so we decided to assign the copies a unique barcode. This allows us to easily keep track of the individual copies of the books.

When it came to model the concept of the book copies, we had three options: to have the book copy as a separate normal entity, to store the copies as multi-valued attribute of the Book or as a weak entity dependent on the Book entity. The reason why we chose the first option is because unlike a typical weak entity the book copy has a barcode that can be used to uniquely identity it. In addition, the book copy is tightly related to a book so it could not stand on its own as an entity and we did not want to facilitate a situation where copies of a book that does not exist are stored in the system. Therefore, we chose to add it as a multiple-valued attribute of the Book entity.

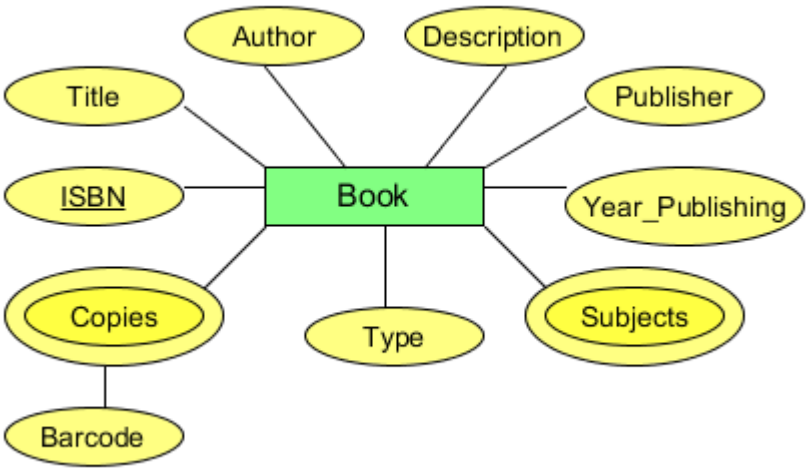

Figure 3: The Book entity in the ER diagram

The **Map** entity is also included, as it is mentioned in the requirements. The key attribute is the ISBN, which is allowed because ISBNs can be assigned to maps as well [1]. The entity has the following simple attributes Title, Author, Description, Publisher, Year_Publishing and No_Copies. No_Copies is added because we assumed that the library would like to store of the number of copies of each map it has. We are not concerned with tracking the individual copies of the maps in the system because the library does not loan out maps. We assumed that although maps are not loanable, the library would like to store information about them in the system.

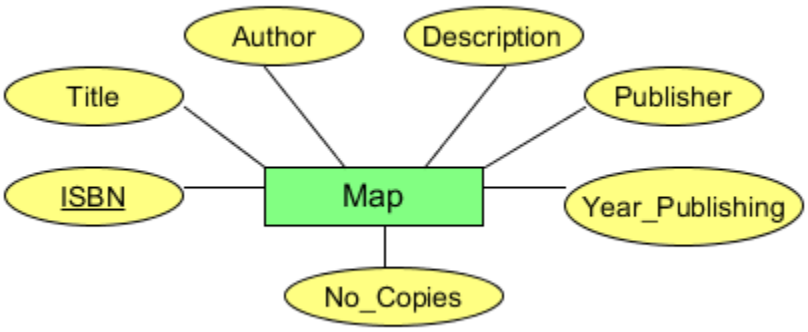

Figure 4: The Map entity in the ER diagram

Relationships

In our ER diagram, two important relationships between entities, have been identified. The first one is a binary relationship called **Uses**. This relationship between the Person and Card entity has the following structural constraints: a 1:1 (one-to-one) cardinality and a total participation constraint. This means that exactly one member of Person set, and one member of the Card set are required to take part in the Uses relationship. The total participation constraint implies that both members must be fully involved in the relationship, and that the relationship does not hold if any of them are missing.

The next relationship is called **To_Borrow**. This is a binary relationship between the Card and Book entity. The To_Borrow relationship has a N:M (many-to-many) cardinality and a partial participation constraint, which basically means that many cards could borrow multiple books, but they don't have to. Unlike the Uses relationship, To_Borrow has two simple attributes associated with it. The first one is called Is_Returned and it signifies the status of a book borrow, whether the loaned item has been returned or not. The Date_Borrowed attribute stores the date on which a book borrow has taken place.

We have not added a relationship like the To_Borrow one, between the Card and Map entities, because in the case, the library states that it does not loan out maps.

However, this diagram can be improved by converting it into an Enhanced Entity Relationship (EER) diagram.

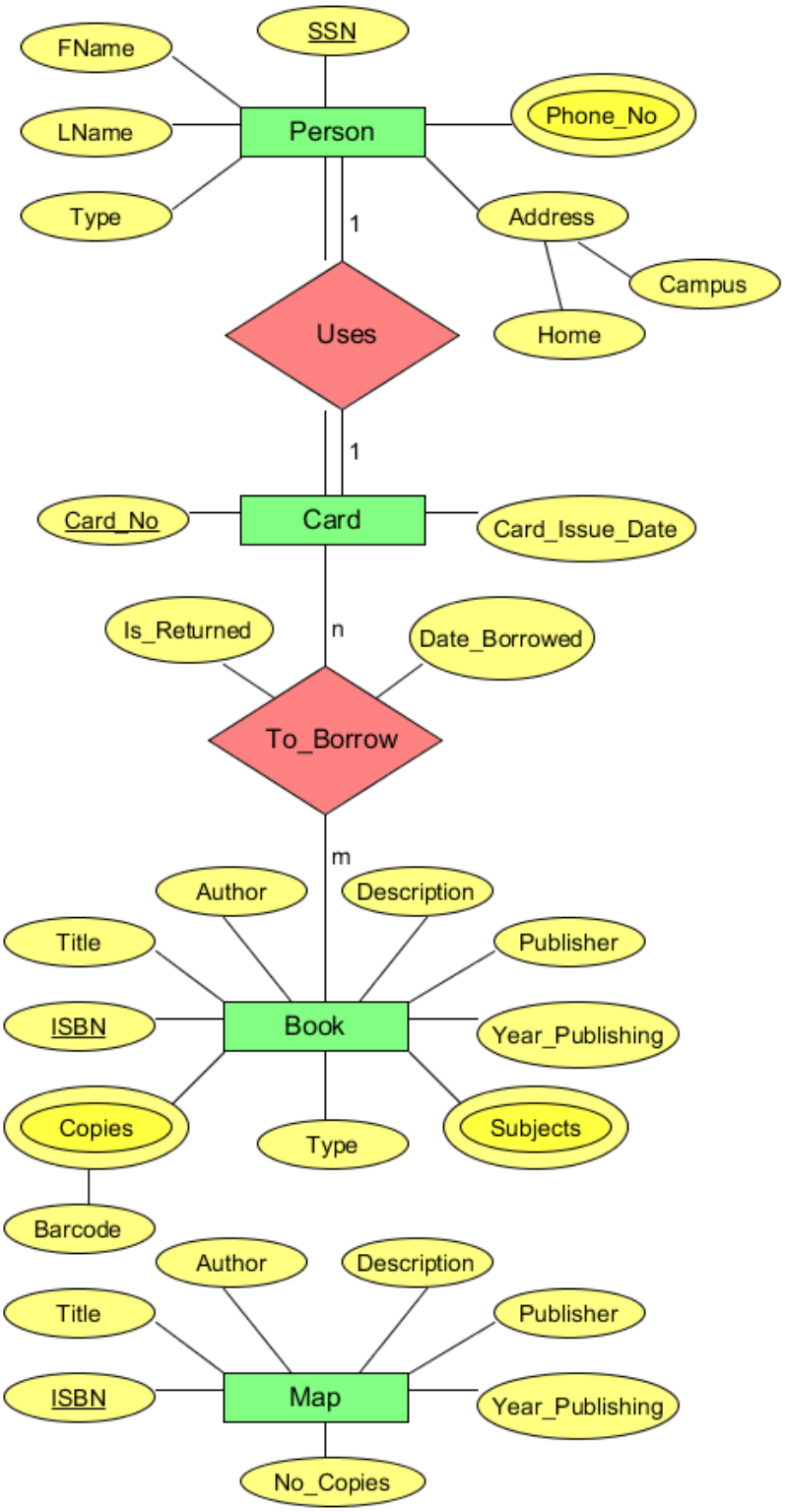

Figure 5: Entity-Relationship (ER) diagram

EER diagram

The EER diagram has several additions and differences compared to the ER diagram.

We start by describing the changes made to the Person entity. The terms library member and librarian are mentioned in the case. The way we interpreted the case requirements, the librarians are the ones that use the library loaning system to borrow books to members of the library, which can be either students or professors but not both at the same time. The Person entity then becomes a superclass and two new entities are added, Librarian and Member, which are its subclasses. This hierarchy has a disjoint constraint, noted with the letter 'd', because as stated before, Librarian and Member are specialisations of the Person entity. This superclass entity also has all the attributes that are common to its subclasses, which are the same ones as the ER version of the Person entity, except for the Type attribute which was removed. The Librarian subclass has an attribute called Position, which is used to store the job title that the librarian have in the library such as chief librarian, reference librarian, assistant librarian etc. The Member subclass has a simple attribute called Role which stands for student, professor or other. We decided that it would be necessary to distinguish between library members that are students or professors, because the library has different business rules regarding each one. The type of disjoint is total which means that a member of the Person set must be either a Librarian or a (library) Member in this context. Going forward, we shortly discuss the Uses relationship. We have kept the relationship between the Person and Card entity unchanged. Our reasons for doing so, is because although Person specialises into a Librarian or Member, we assumed that librarians can also use library cards to borrow books.

Moving on, we added the Item entity as a superclass for the subclasses Book and Map. The Item holds the following common attributes: ISBN, Title, Author, Description, Publisher and Year_Publishing, of which ISBN is the key attribute. This hierarchy has a total disjoint constraint, which means that an Item must either be a Book or a Map, but not both. The Map subclass has the simple attribute No_Copies. The Book subclass has the multiple Subjects, Copies with the simple attribute Barcode, and the simple attribute Type which could be normal, reference and rare. The reason why we have not specialised the Book entity into a Normal_Book, Rare_Book and Reference_Book is because we could not find any attributes for those possible subclasses. Therefore, we made the design choice to simply store the Type of a book as an attribute on the Book entity. We kept the To_Borrow relationship the same as in the ER diagram, because the case mentions that only a certain type of book can be borrowed.

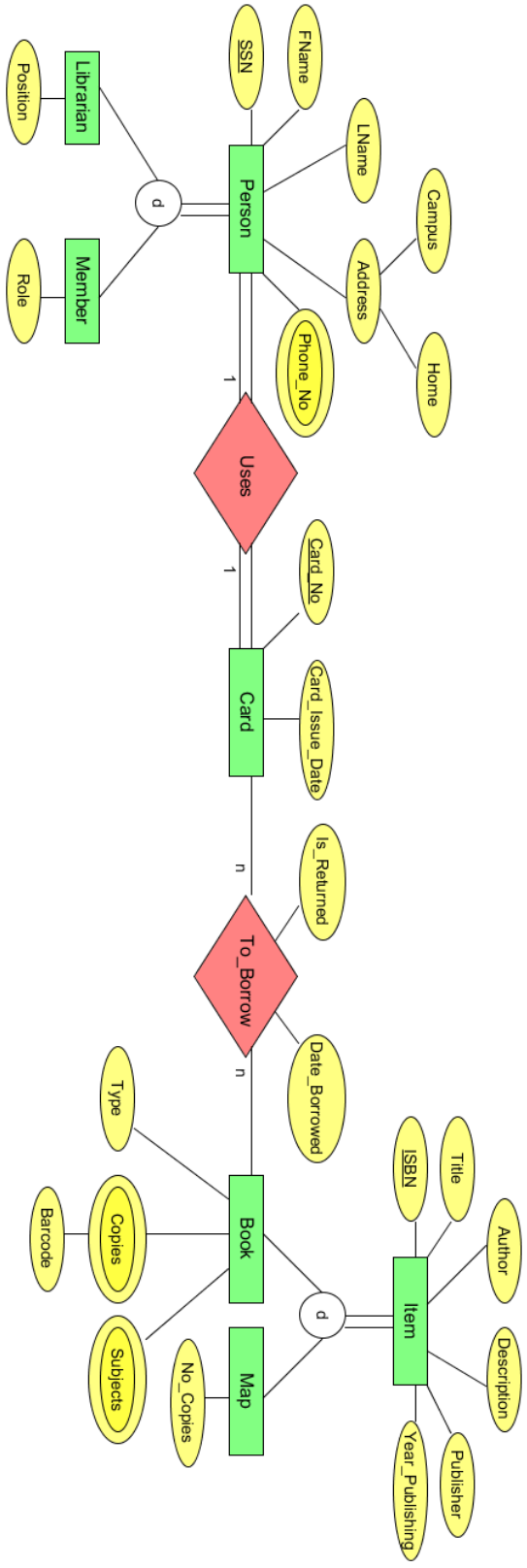

Figure 6: Enhanced-Entity Relationship (EER) diagram

Table Design

Choosing a database type

When it came to choose a database type for our database solution, we had three options available:

- Relational database
- NoSQL database
- Hybrid database

We chose a relational database for our database solution for several reasons. First, we considered our experience and skill set as developers, and so we decided that working with a familiar technology stack and SQL, is something we prefer a lot. The learning curve required to mastering a NoSQL database such as the document-based database called MongoDB, would take too much time out of the total time allocated to this project. Secondly, from our analysis of the requirements, we think that the data entities have many relations between them, so in this case the relational database would be the best fit. In the end, we chose to implement a relational database, with MSSQL as the RDMS (Relational Database Management System).

In addition, we strongly consider the option of switching to a hybrid database in the future for this project. We predict that the data in some tables could grow at a high rate, enough to qualify for a Big Data solution. Therefore, a relational and a NoSQL database solution, would meet the needs of the project if such a case would manifest in the future.

Relational model

The next step is to move on to the design of GTL relational model. To accomplish this, we followed the steps of the ER-to-Relational Mapping algorithm and then the steps for mapping EER constructs to relations.

In total, there are four options for mapping specialisation, of which we chose option 8A that states “Multiple relations-Superclass and relations” [2]. This rule can be applied to any specialisation and in our case, resulted in the creation of three relations: Person, Librarian and Member for the first specialisation Person and for the second one also three relations: Item, Book and Map. Other options that could have been chosen by us are 8B and 8C.

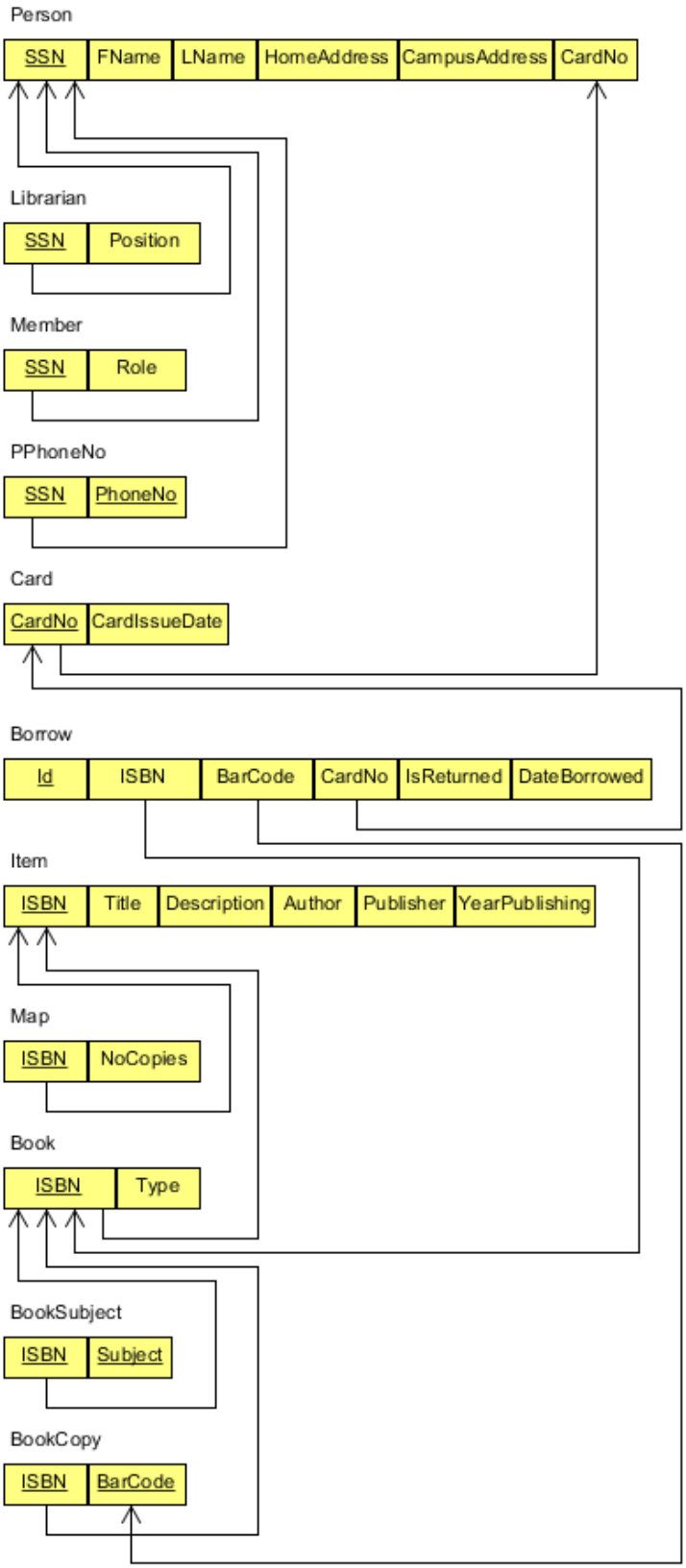

Figure 7: Relational database schema

Normalisation

We studied our relational schema to try to figure out what functional dependencies there are in the tables and in what normal form they each are. We concluded that all the tables are in 3NF and BCNF. Therefore, we did not make any adjustments to the table design.

Data definition language (DDL) statements

These are the DDL statements that we used to create our tables and relations in MSSQL, starting from the top left to the bottom right.

```
USE GTL
CREATE TABLE LibraryCard (
    CardNo NUMERIC(20) NOT NULL IDENTITY(1,1),
    IssueDate DATE NOT NULL,
    PRIMARY KEY (CardNo)
);
CREATE TABLE Person(
    SSN NUMERIC(20) NOT NULL,
    FName VARCHAR(50) NOT NULL,
    LName VARCHAR(50) NOT NULL,
    HomeAddress VARCHAR(75),
    CampusAddress VARCHAR(75),
    CardNo NUMERIC(20) NOT NULL,
    PRIMARY KEY (SSN),
    FOREIGN KEY (CardNo) REFERENCES LibraryCard(CardNo) ON DELETE CASCADE
);
CREATE TABLE Librarian(
    SSN NUMERIC(20) NOT NULL,
    Position VARCHAR(50) NOT NULL,
    PRIMARY KEY (SSN),
    FOREIGN KEY (SSN) REFERENCES Person(SSN) ON DELETE CASCADE
);
CREATE TABLE Member(
    SSN NUMERIC(20) NOT NULL,
    Personification VARCHAR(50) NOT NULL,
    PRIMARY KEY (SSN),
    FOREIGN KEY (SSN) REFERENCES Person(SSN) ON DELETE CASCADE
);
CREATE TABLE PhoneNo(
    SSN NUMERIC(20) NOT NULL,
    Number INT NOT NULL,
    PRIMARY KEY (SSN, Number),
    FOREIGN KEY (SSN) REFERENCES Person(SSN) ON DELETE CASCADE
);
CREATE TABLE Item(
    ISBN NUMERIC(13) NOT NULL,
    Title VARCHAR(50) NOT NULL,
    Author VARCHAR(50),
    ItemDescription VARCHAR(50) NOT NULL,
    Publisher VARCHAR(50),
    YearPublishing INT,
    PRIMARY KEY (ISBN)
);
CREATE TABLE Map (
    ISBN NUMERIC(13) NOT NULL,
    NoCopies INT,
    PRIMARY KEY (ISBN),
    FOREIGN KEY (ISBN) REFERENCES Item(ISBN) ON DELETE CASCADE
);
CREATE TABLE Book (
    ISBN NUMERIC(13) NOT NULL,
    BookType VARCHAR(50) NOT NULL,
    PRIMARY KEY (ISBN),
    FOREIGN KEY (ISBN) REFERENCES Item(ISBN) ON DELETE CASCADE
);
CREATE TABLE BookCopy (
    ISBN NUMERIC(13) NOT NULL,
    Barcode NUMERIC(20) NOT NULL,
    PRIMARY KEY (ISBN, Barcode),
    FOREIGN KEY (ISBN) REFERENCES Book(ISBN) ON DELETE CASCADE
);
CREATE TABLE BookSubject (
    ISBN NUMERIC(13) NOT NULL,
    BookSubject VARCHAR(50) NOT NULL,
    PRIMARY KEY (ISBN, BookSubject),
    FOREIGN KEY (ISBN) REFERENCES Book(ISBN) ON DELETE CASCADE
);
CREATE TABLE Borrow (
    Id NUMERIC(20) NOT NULL IDENTITY(1,1),
    ISBN NUMERIC(13) NOT NULL,
    Barcode NUMERIC(20) NOT NULL,
    CardNo NUMERIC(20) NOT NULL,
    IsReturned BIT NOT NULL,
    DateBorrowed DATE NOT NULL,
    PRIMARY KEY (Id),
    FOREIGN KEY (ISBN, Barcode) REFERENCES BookCopy(ISBN, Barcode) ON DELETE CASCADE,
    FOREIGN KEY (CardNo) REFERENCES LibraryCard(CardNo) ON DELETE CASCADE
);
```

Figure 8: DDL statements used to create the database tables

Below, we have included the auto-generated database diagram that was created as a result of the DDL statements.

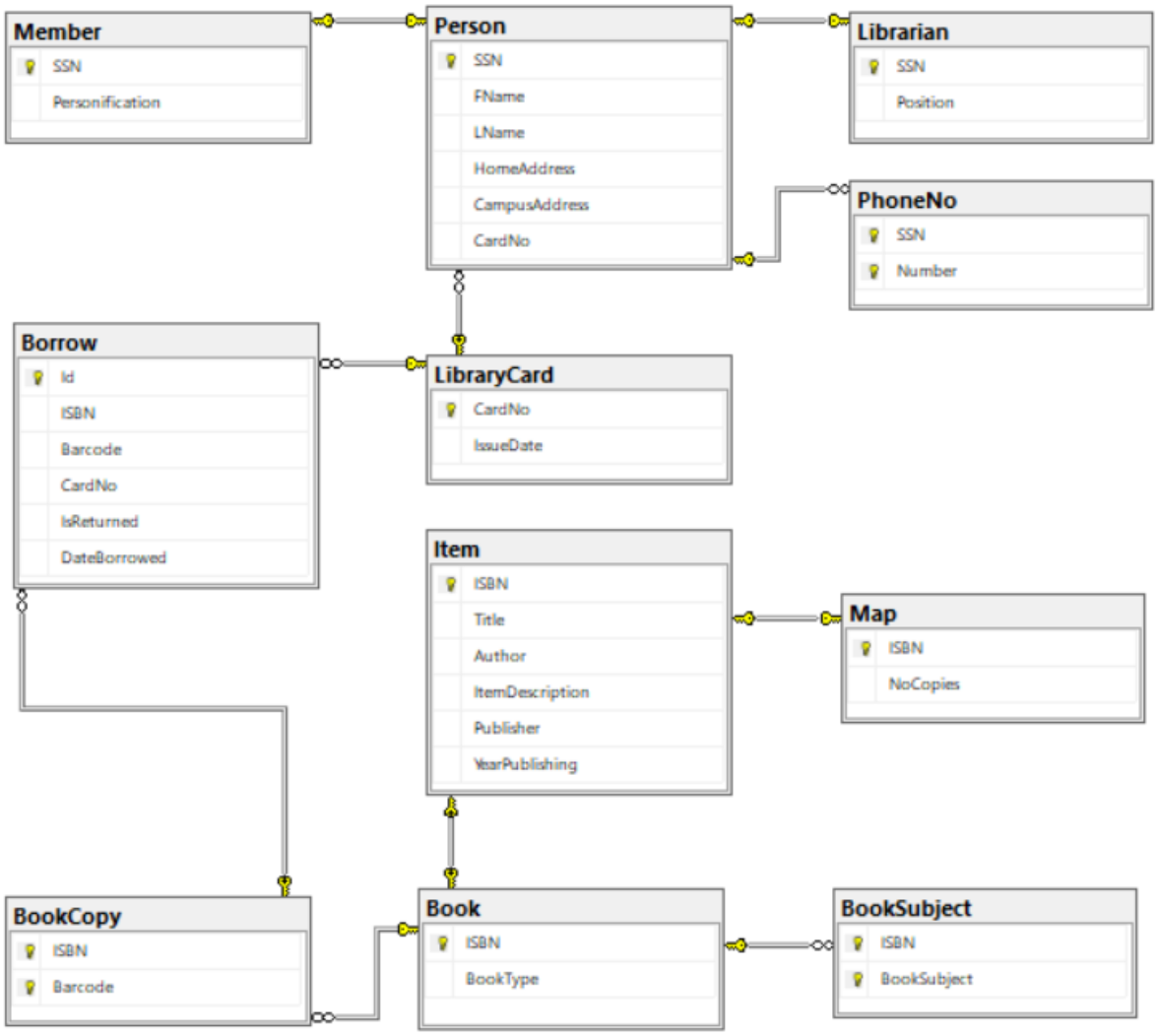

Figure 9: GTL Relational Database Diagram

Data generation

We used a data generation tool called SQL Data Generator 4 provided by Red Gate Software Ltd. We inserted a volume of 50.000 data rows into our most important tables such as the Borrow table. Unfortunately, the data generation operation did not enforce our established business rules, because we did not add certain constraints. For example, the tool inserted books that are not of the type 'Normal', into the Borrow table because we did not set a constraint, and the same copy of the same book was loaned by two different persons on the same day.

Query Design

In this part, we describe five of the most useful queries, that we identified in this case.

Insert loan query

The loan query starts by initialising the parameters, then executes 5 procedures that verifies that the member has loaned less than 5 books, the book is loanable, the book copy exists, the card exists, and the copy is available. Then it checks if all the procedures return true, if all of them are true it starts the query that inserts the loan into the loan table, if any of them are false it prints “CANNOT INSERT LOAN!”.

```
USE GTL
DECLARE @ISBN NUMERIC(13) = '19190630000',
    @BARCODE NUMERIC(20) = '913719133000000',
    @CARDNO INT = '2',
    @ret_value_BookLimit BIT,
    @ret_val_IsLoanable BIT,
    @ret_val_BookCopy BIT,
    @ret_val_ValidCard BIT,
    @ret_val_CopyAvailable BIT;
EXEC    [dbo].[VerifyBookLimit]
        @cardno = @CARDNO,
        @loanable = @ret_value_BookLimit OUTPUT
EXEC    [dbo].[VerifyIsLoanable]
        @isbn = @ISBN,
        @loanable = @ret_val_IsLoanable OUTPUT
EXEC    [dbo].[VerifyBookCopy]
        @isbn = @ISBN,
        @barcode = @BARCODE,
        @exists = @ret_val_BookCopy OUTPUT
EXEC    [dbo].[VerifyCard]
        @cardno = @CARDNO,
        @exists = @ret_val_ValidCard OUTPUT
EXEC [dbo].[VerifyCopyAvailable]
    @barcode = @BARCODE,
    @available = @ret_val_CopyAvailable OUTPUT
IF(@ret_val_BookCopy = '1' AND @ret_val_CopyAvailable = '1' AND
    @ret_val_IsLoanable = '1' AND @ret_val_ValidCard = '1' AND @ret_value_BookLimit = '1')
BEGIN
    INSERT INTO Borrow (ISBN, Barcode, CardNo, IsReturned, DateBorrowed)
    VALUES (@ISBN, @BARCODE, @CARDNO, 'false', GETDATE())
    PRINT 'LOAN INSERTED SUCCESFULLY!'
END
ELSE
BEGIN
    PRINT 'CANNOT INSERT LOAN!'
END
```

Figure 10: Insert loan query

Look up book by title query

This query selects the information from item, book, and book subject with the title chosen. It selects the book and subject based on the ISBN found when searching for a book and uses an inner join on the three tables.

```
USE GTL
SELECT I.ISBN, I.Title, I.Author, I.ItemDescription, I.Publisher, I.YearPublishing, B.BookType, S.BookSubject
FROM (Item AS I JOIN Book AS B ON B.ISBN = I.ISBN JOIN BookSubject AS S ON S.ISBN = B.ISBN)
WHERE I.Title = 'LemonChiffon Nepal Isaac';
```

Figure 11: Look up book by title query

View currently on loan query

This query selects the information from borrow, item and person from the books that are not returned in borrow. It selects the item based on the ISBN from all the rows that are returned, the person based on the card number from all the rows that are returned and uses an inner join on the three tables.

```
USE GTL
SELECT I.ISBN, I.Title, Bo.Barcode, Bo.DateBorrowed, P.FName, P.LName
FROM (Borrow AS Bo JOIN Item AS I ON Bo.ISBN = I.ISBN JOIN Person AS P ON Bo.CardNo = P.CardNo)
WHERE Bo.IsReturned = 'false';
```

Figure 12: View currently on loan query

View not loanable query

This query selects the information from item where the ISBN is the same as the ISBN from all the books with the book type “normal”. It uses a nested loop to find all the books with the book type “normal” and returns the rows.

```
USE GTL
DECLARE @StartTime datetime, @EndTime datetime
SELECT @StartTime = GETDATE()
SELECT I.ISBN, I.Title, I.Author, I.ItemDescription, I.Publisher, I.YearPublishing
FROM Item AS I
WHERE I.ISBN NOT IN ( SELECT B.ISBN
                      FROM Book AS B
                      WHERE B.BookType = 'Normal');
SELECT @EndTime = GETDATE()
SELECT DATEDIFF(ms, @StartTime, @EndTime) AS [Duration in microseconds]
```

Figure 13: View not loanable query

View overdue books query

This query selects the information from borrow, item and person from the books that are not returned in borrow. It uses a nested loop to select the library cards of the persons, it then uses another nested loop to select the members based on the SSN, lastly it uses an OR statement to decide if the members personification is a student or a professor. It gives different borrow time based on if you are a student or a professor.

```
USE GTL
SELECT Bo.DateBorrowed, I.Title, P.Fname + ' ' + P.LName AS 'Full Name', DATEDIFF(DAY, Bo.DateBorrowed, GETDATE()) AS 'Days Overdue'
FROM (Borrow AS Bo JOIN Item AS I ON Bo.ISBN = I.ISBN JOIN Person AS P ON Bo.CardNo = P.CardNo)
WHERE Bo.CardNo IN ( SELECT P.CardNo
                     FROM Person as P
                     WHERE (P.SSN IN (SELECT M.SSN
                                      FROM Member AS M
                                      WHERE (M.Personification = 'Student' AND Bo.IsReturned = 'false' AND (DATEDIFF(DAY, Bo.DateBorrowed, GETDATE()) >= 21)
                                      OR
                                      (M.Personification = 'Professor' AND Bo.IsReturned = 'false' AND (DATEDIFF(MONTH, Bo.DateBorrowed, GETDATE()) >= 3))))));
```

Figure 14: View overdue books query

Important Indexes

Clustered indexes

Clustered indexes use one column as the index which it is sorted by called the key value, it can only be one clustered index per table as you can sort a table by one column only. A table is sorted in an order when it contains a clustered index, if a table do not contain a clustered index, it is called a heap which is an unordered structure. A clustered index is automatically added when a primary key is added, a relational database in the second normal form or higher will contain a clustered index [3].

Non-clustered indexes

A non-clustered index contains the non-clustered index key values and each key value entry has a pointer to the data row that contains the key value. The pointer is called a row locator, there are two types of structures, one for the heap and one for clustered table. For the heap, the row locator points to the row. For a clustered table, it points to the clustered index key [3].

Expected improvement

The estimated execution plan showed that we could create indexes for all out queries except 'View overdue books'. The impact on these four queries where, 'View currently on loan' (8.67), 'View not loanable' (14.83), 'Look up book by title' (99.27) and 'Insert loan' (98.74). Based on these estimations, we decided to implement the recommended non-clustered indexes for 'View not loanable', 'Look up book by title' and 'Insert loan'. This should improve our system drastically. We decided on not adding a non-clustered index for 'View currently on loan' as it only has an impact of 8.67 and is not a query that will be executed a lot, so it is not deemed valuable to use the resources to create a non-clustered index for it.

```
USE GTL
CREATE NONCLUSTERED INDEX Borrow_Card_IsReturned ON Borrow (CardNo DESC, IsReturned ASC);
CREATE NONCLUSTERED INDEX Item_Title ON Item (Title DESC);
CREATE NONCLUSTERED INDEX Borrow_Barcode_IsReturned ON Borrow (Barcode DESC, IsReturned ASC);
CREATE NONCLUSTERED INDEX Book_Booktype ON Book (BookType DESC);
```

Figure 15: SQL statements for creating non-clustered indexes

Based on the estimated execution plan, we expected the 'Insert loan' to go from 15ms to 2ms, 'Look up book title' to go from 13ms to 2ms, 'View currently on loan' to not change, 'View not loanable' to go from 900ms to 755ms and 'View overdue books' to not change. As seen in Figure 16, you can see 'Insert loan' and 'Look up book by title' got reduced by approximately the same as the estimated high 90%. While 'View not loanable' only got reduced by only 8% when it was estimated to be reduced by 14.83%. Surprisingly, both 'View currently on loan' and 'View overdue books' got reduced without adding non-clustered indexes for them, 'View currently on loan' got reduced 1/5 of the time (20%) and 'View overdue books' got reduced by over 2/5 of the time (43%).

Query name	Timing before (ms)	Timing after (ms)	Difference (ms)	Percentage saved
Insert Loan	15	0	15	100%
Look up book by title	13	0	13	100%
View currently on loan	220	175	45	20%
View not loanable	900	830	70	8%
View overdue books	210	120	90	43%

Figure 16: Table displaying the time difference before and after adding the indexes

Difference in execution plans for the 'Look up by title' query

In Figure 17, it starts with select (Cost: 0%), goes into the nested loops inner join (Cost: 0%), then goes into another nested loops with inner join (Cost: 4%), then does a clustered index scan on Item with primary key (Cost: 96%), goes into another clustered index seek on BookSubject with primary key (Cost: 0%), finally it does a clustered index seek on Book with primary key (Cost: 0%).

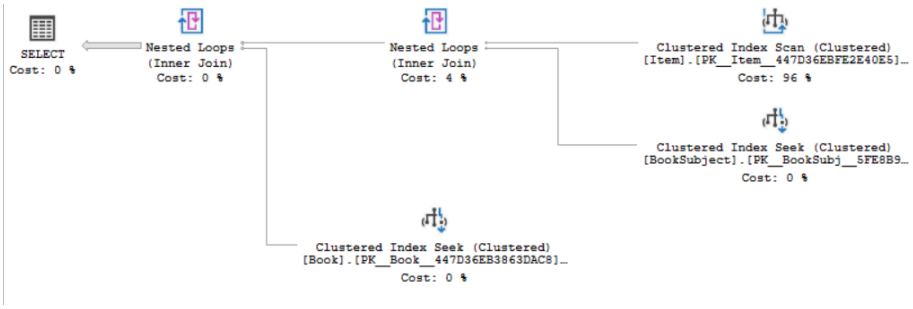

Figure 17: 'Look up by title' Execution Plan before adding the indexes

In Figure 18, it starts with select (Cost: 0%), goes into three nested loops inner join (Cost: 0%), then does a non-cluster index seek on Item wit Title (24%), then uses the key from the non-clustered seek to look up on the table Item (24%), goes into clustered index seek on BookSubject with primary key (Cost: 24%), finally it does a clustered index seek on Book with primary key (Cost: 28%).

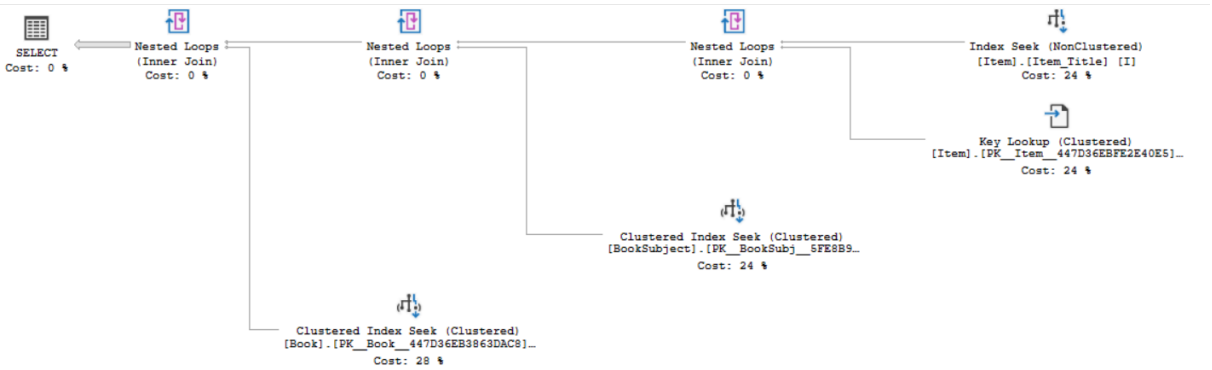

Figure 18: 'Look up by title' Execution Plan after adding the indexes

Comparing these two execution plans, it shows adding non-clustered keys save time, before adding the non-clustered indexes, majority of the resources, 96% was used on the clustered index scan on Item with the primary key that used and the rest was used on the second nested loops before the scan.

After adding the indexes, one more loop was added and it found the Item based on the title instead of on the primary key, it now only uses 24% to find the title with the non-clustered index seek and 24% on the clustered key lookup. Adding the indexes also allowed the query to use its resources on the clustered index seek for BookSubject and Book.

Databases & Security

There are two types of security that can be added to the database, authentication and authorisation.

Authentication

Authentication is used to authenticate that a user has access to the server by submitting their credentials. The server evaluates if the credentials allow access and establishes the identity.

There are two authentication modes, windows authentication and mixed mode. Windows authentication is the default and uses windows as the authenticator. Mixed mode uses both windows authentication and SQL server authentication, the username and password are stored in the database rather than on windows.

In production we are using windows authentication as we have the software and the database on the same computer and are using LocalDB. When we will deploy the software, it will use mixed mode because the database will not be on the same computer as the software and will have multiple users that are not windows accounts. There will be users of the system where we will store a username and a hashed/salted password in the database for authentication [4].

Authorisation

A role is used to restrict access and control from users, it uses the role to know what the user is authorised to do, what table the user can interact with and what they can do to those tables. You can give the public role which the permissions that every role will be given, as everyone inherits from this role, it cannot be dropped, added to nor deleted from.

In our system we need to restrict the users (librarians) based on what type of librarian they are. We have five different roles that are needed for the library, Chief librarian, departmental associate librarian, reference librarian, check-out staff and library assistant.

We would use the principle of least privilege to decide what permission(s) each role would be given. As an example, we would give the chief librarian role the permission to add, update, read and delete rows in all tables, they would not be allowed to change or drop the tables. The check-out staff would only be allowed to add to the borrow table [4]

Database Programming

In MSSQL, we have the option of adding programming directly to the database. We have the possibility of using stored procedures, functions or triggers to enforce the business rules of the case. We created multiple stored procedures, which were used to verify whether a loan can or cannot be inserted into the Borrow table. The procedures are quite simple and to the point so we will only include and discuss two of the totals of five.

The VerifyIsLoanable stored procedure

This stored procedure takes the ISBN of a book and verifies whether the type of the book is 'Normal'. According to the business rules of the library, only books of the 'Normal' type can be loaned. The procedure has two parameters, one for the ISBN of the book that we need to check and one that is returned as an output, and where we store the result of the procedure.

```
ALTER PROCEDURE [dbo].[VerifyIsLoanable](
    @isbn NUMERIC (13),
    @loanable BIT OUTPUT
) AS
BEGIN
    IF EXISTS (
        SELECT ISBN, BookType
        FROM Book
        WHERE ISBN = @isbn AND BookType = 'Normal')
        BEGIN
        SELECT @loanable = 'true';
        END
END;
```

Figure 19: VerifyIsLoanable stored procedure

The VerifyBookLimit stored procedure

This stored procedure applies the business logic for checking that a holder of a card can only loan a maximum of five books, which are not returned.

```
ALTER PROCEDURE [dbo].[VerifyBookLimit](
    @cardno NUMERIC (20),
    @loanable BIT OUTPUT
) AS
BEGIN
    IF EXISTS (
        SELECT CardNo
        FROM Borrow
        WHERE CardNo = @cardno AND IsReturned = 'false')
        BEGIN
        SELECT *
        FROM Borrow
        WHERE CardNo = @cardno AND IsReturned = 'false';
        IF (@@ROWCOUNT < 5 )
        BEGIN
        SELECT @loanable = 'true'
        END
        END
END;
```

Figure 20: VerifyBookLimit stored procedure

Conclusion

In the end, the process of developing our database solution was a valuable learning experience. At the start, we were already familiar with the environment of MSSQL and relational databases. By the end of this project, we managed to investigate these topics more deeply and gain a better understanding of them. Even though, we did not utilise them, we also learned about NoSQL databases and how to choose the right database type for a project.

References

[1] FAQs: ISBN Eligibility | ISBN.org, 2014, [Online]. Available: https://www.isbn.org/faqs_isbn_eligibility

[2] Ramez Elmasri, Shamkant B. Navathe, "Fundamentals of Database Systems (7th Edition)", Pearson, 2015, ISBN: 978-0133970777

[3] Clustered and Nonclustered Indexes Described, 2019, [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-2017

[4] Overview of SQL Server Security, 2017, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/overview-of-sql-server-security

Test

Introduction

In this report, we will introduce and discuss the Georgia Tech Library (GTL) case and we will present our proposed solution for the case with a strong focus on the software testing aspect of it. In continuation, we will describe the development process that was followed by us, how testing fit in with it and the various testing techniques that were used. We plan to go through the evolution of the project and how we managed to adapt the testing process to fit our changing environment. We explain our motivations and reasons for why we chose to test certain parts of the system and how we decided which parts were important enough to need focused or additional testing.

Description of the GTL case

The main point of the GTL case that we were tasked with, is that the library needs a book loaning system and that we will build it for them. There are several business rules that were mentioned in the case, that must be implemented and accounted for. As well as that, the case gives us an overview of the library and how it functions. At the end of the case, we are assigned with building a working prototype for managing the library and its areas of concern. The case does not specify whether the library already has a software system like the one they are requesting us to build except for the mention of an online catalog of books which is quite vague. Therefore, we assumed that we would need to build a completely new system for them.

Project Initiation Document

For the sake of this project, we constructed a Project Initiation Document (PID). The purpose of a PID is to clearly define what the library, which is the customer in this case, should expect to receive from us, the suppliers of the system, at the end of the project. The document is included below.

Business objective

- To modernise the library's loaning system by switching to a digital solution

Project objective

- To implement a working prototype for a library loaning system that will keep track of the members, books and borrowing activity

Scope

- Boundaries
    - Member registration
    - Book management
    - Tracking of items status such as available or borrowed / on loan
    - Reliable item loaning functionality
    - Sending notifications according to member type

    - Enforcing the communicated business rules such as
        - Restricting the borrowing of certain items
        - Restricting the number of books, a member can loan
- Activities
    - Supplier will only provide the software of the discussed solution; additional software that may be required as a platform for the solution to run on is not provided;
    - Supplier will not provide any hardware components; that is considered the customers responsibility
    - Supplier will not be responsible for any security issues of the system
    - Supplier will not be taking on the task of maintaining the system, after the completion of the project
    - Supplier will not be tasked with training the library's employees on how to use the system
- Deliverables
    - Software system consisting of a working prototype with the functionalities that were agreed upon by both parties
    - Documentation about the systems design, implementation, usability and other relevant features

Constraints

- The development approach is an Agile one, based on SCRUM
- The proposed software system will run on the Windows 10 operating system
- The technology stack used is the .NET framework with the C# programming language

Authority

- The person responsible for representing the customers interests and tasked with approving the final product is one of the developers

Resources

- The projects cost is fixed
- The staff working on developing the system consists of two developers
- Time to deliver the solution is 6 weeks
- The hardware required to develop the system is provided by the supplier

Besides the specifics of the product that is to be delivered, the PID also describes the project regarding important constraints such as project scope, delivery time, the desired quality and the cost of the project. These factors were taken into consideration by us, when we chose our test strategy and test approach.

Risk analysis

Before beginning any project, it is critical to conduct a risk analysis to be able to handle any possible issues that might force the project to a halt. To be thorough, we decided to look at both the project and the product risks in this analysis.

Project risks

No.	Risk	Description	Impact	Priority	Mitigation
1	Lack of testing experience	Team doesn't have enough concrete experience with the process of thoroughly testing a software system	High	1	Team must rely on following best practices and standards instead of experience
2	Lack of testing tools knowledge	Team is unacquainted with the testing tools necessary to properly test the system	High	3	Team must allocate time resources to reduce knowledge gaps
3	Product owners not available	The actual product owners cannot be involved in the process which is an important factor in agile development	High	2	Team must assign one of its members to act as a product owner for the duration of the project

Product risks

No.	Risk	Description	Impact	Priority	Mitigation
1	Vague or incomplete requirements	The case, from which the requirements are to be extracted, is not detailed enough	High	1	Team must make assumptions when necessary and try to guess what the library would want
2	Lack of time	Team is pressured for time because of the tight deadline	High	3	Team must prioritise and plan their activities in advance
3	Lack of business case	No business case has been provided which makes it hard to figure out what the main objectives that the system should solve are	Medium	2	Team must resign to select some possible objectives from the description of the case

Test strategy

The test strategy is an important key in the success of any project. For our project, we decided that a risk-based analytical strategy would be best suited. Some of the factors that we considered when we picked our strategy are: risk, skills, objectives and product. Managing both project and product risks has been a major concern for us since the beginning. We conducted a detailed risk analysis, where we weigh in on all the risks that we thought we might face. In addition, we also devised ways to mitigate those risks. A risk-based strategy, for us, means that we use testing as a tool to manage the risks. For example, when it comes to the risk caused by a lack of time resources. In this case, we should strive to choose types of tests that allows us to deliver the most value in the small time slot that we have.

Skills, especially testing skills, have also been an obstacle for us during this process. It is necessary to consider the skillset within our team, when deciding on a strategy, to make sure that we can properly implement it in the end. If we take into consideration our limited testing skills, then a strategy such as a standard-compliant one, could have also been a good option for our team.

Our product, which is a small library book loaning system has been a huge factor for deciding our strategy. The product that we deliver to the customer is dependent on the scope of the project that we set out to implement. The project scope has been clearly laid out in the PID document. Besides, the actual features that we need to build, it helps to know the type of the product as well. Because, we know that a prototype for a library system does not have as high of a criticality as a hospital management system for example. This tells us that we don't need a methodical strategy, which implies that we follow an industry-standard for testing.

We continue by giving a definition of the test process that we intend to follow. First, we create test cases based on the acceptance criteria noted on our user stories. In the case of the most critical ones, we might require additional details. Then, we consider the levels at which testing will occur and our levels of automation. Another, step in this process is define our exit criteria or when we are done testing.

In the end, our strategy requires us to frequently use our analytical skills to prevent and manage various risks and to ensure that adequate testing is being carried out.

Test Plan

As stated in the GTL PID, we follow an agile development process along with the SCRUM agile method. The purpose of a test plan is to plan testing activities in advance, to increase the likelihood that they will take place when they should or when we decide to execute certain tests. Like many other types of documentation, when it comes to agile, test plans are a controversial topic. We think that there are valid arguments both for using and for not a test plan. We have chosen to write a test plan for our agile project, mainly for the reason mentioned earlier, regarding their use. Secondly, we think that we can make a test plan work in our case, if we add a few adjustments to it. First, we need to change our mindset and view this document as more of a general guideline for our project tests instead a rigid plan. Secondly, we decided that the plan should only include elements that are helpful in our case.

The contents of our test plan are as follows:

1) Scope of testing
2) Test objectives
3) Test approach
4) Test automation
5) Test coverage
6) Test environment
7) Test tools
8) Test effort and schedule

Scope of testing

The project we will be working on is a software development project which will implement a library loaning system meant to replace GTL's already existing analog loaning system. The system will be used internally, and our testing will be focused on the Business Layer. The main part is focused on the loaning of books to its members. The project objectives we are concerned with are keeping track of the loans, the members and the due dates.

Test objectives

Our primary test objective is to find defects in the solution that we implemented. Based on the requirements of the GTL case, we also identified several other test objectives. Their purpose is to help us ensure the most critical functionalities of our system will be tested. They are as follows:

1) Verify that a loan is registered in the system
2) Verify that system does not allow a member to borrow more than a maximum of five books simultaneously
3) Verify that system does not loan out books that can't be borrowed
4) Verify that system does not allow members with cards older than four years to borrow books
5) Verify that a loan's status is updated when member returns the book

Test approach

The test approach is described as the actual implementation of a test strategy. Test strategies can be reused across multiple projects, while testing approaches must be tailored to each individual project. At this point we need to introduce the Agile Testing Quadrants [1]. This is a tool used to help developers cover all the types of tests could be considered useful in the context of their project. The tests listed in each quadrant have a different purpose. For example, the types of tests in Q1 are unit and component tests. These tests have the following characteristics: support the team, are technology-facing and automated. We will use Test Driven Development (TDD) to implement Q1 tests. This is where we will focus most of our testing resources and efforts. Another type of test we are interested in are functional tests which are in Q2. These tests are supporting the team, business-facing and automated or manual. We decided that in the context of our project and considering our testing objectives and resources, the tests that we will implement for our solution will be Q1 and Q2 tests.

In terms of the testing techniques we plan to use both static and dynamic ones, however we will mostly focus and document the latter techniques. From the static testing techniques, we intend to use frequent informal reviews to verify our work accordingly. In addition to that, we will also follow a pre-determined coding standard that the team agreed on beforehand. Regarding the dynamic testing, we will mostly utilise white-box techniques.

Regarding the exit criteria for testing, we have chosen meeting the 80% test coverage goal and when the team is satisfied with the effort put in it and they agree to end the testing activities. Another obvious exit criterion for us is when we run out of time resources.

Out of all the test objectives mentioned above, we considered the one loan relating to the loan feature the most important one and we intend to focus our testing efforts into ensuring that this feature is tested throughout.

All the members of our team will be heavily involved in the testing activities and share about the same testing skillset so there will be no specialisation of work on our part.

Test automation

Test scripting or automation is an important concept for agile testing. Automating tests, such as unit and component tests, is a great way to save both time and effort. For this reason, we will strive to automate as many tests as possible, especially tests relating to the loaning feature, which is a critical feature to us as mentioned in the test approach. Another motivation for test automation, is the fact that it supports regression testing, which is very important in agile testing. We used NUnit as a tool for automating tests.

Test coverage

Metrics can be very helpful when testing, to make sure that the team stays on track. For us, the metric that we decided to track, throughout our development process, is test coverage. The standard, that we set out to achieve or maintain, is to have at least 80% of the code covered by tests, at all time. Using TDD helped us tremendously when it came to sticking to our test coverage goals.

We used the in-build Visual Studio tool for measuring the test coverage. Although, we did not log our numbers in a formal document, we made sure to check on our test coverage numbers a few times a day, while working on the code. As a team, we found it was very motivating to ensure that our test coverage met the standard. On the same hand, it also drove us to write more tests, because we were thinking of ways to raise our numbers even more.

Our latest numbers displaying our test coverage, are shown below.

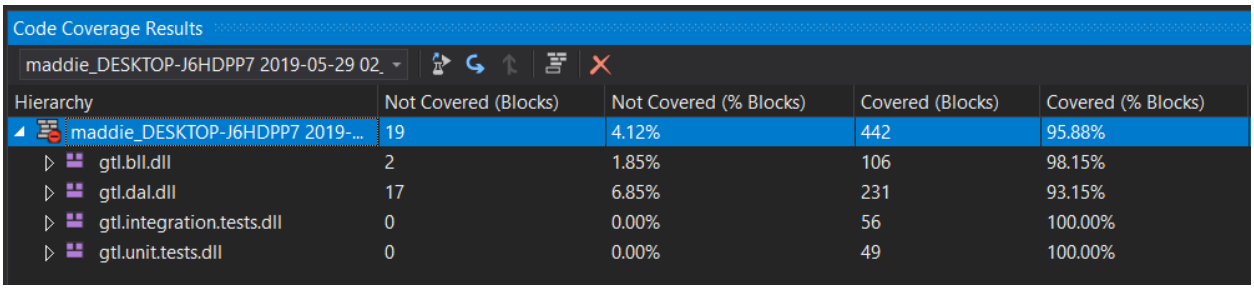

Figure 1: Visual Studio test coverage analysis

Test environment

When it comes to a testing environment, the aim is to have it as similar to the production environment as possible. In an ideal scenario, we would use a Virtual Machine or Docker to ensure a consistent and proper testing environment for our project. Also, we would utilise a remote database to have the testing process be as similar to production mode as possible. However, because of time and cost restrictions, we could not opt for those options. Instead, we used the technology that was already available to us, which includes one laptop running on a Windows 10 operating system and a local MSSSQL database.

Testing tools

We use the following testing tools:

- Unity Container for helping us implement dependency injection and inversion of control
- NUnit for .NET framework for writing and automating unit tests.
- MOQ as a mocking framework
- Visual Studio IDE and its build-in testing tools

Test effort and schedule

The testing tasks described are to be repeated at the beginning of each Sprint, which takes place for two weeks. The planned test effort and schedule is shown below:

Task	**Members**	**Estimate Effort**
Test analysis and design	Developers/Testers	1 day
Test implementation	Developers/Testers	9 days
Test execution and evaluation	Developers/Testers	4 days
Total	Developers/Testers	14 days

Test Management

Managing tests within an agile project is both an interesting and complex topic. The most challenging part of this whole undertaking for us was certainly trying to figure out how to reconcile test management with agile. It is a well-known myth that agile development does not require any documentation. Documenting one's work in a lightweight manner, is encouraged in agile. We agree that having minimalistic documentation that is useful for the team and makes sense in the context of the project, is an overall positive. The issue we faced with test management is that the idea of carefully planning out our tests in advance still seems to go against the core of the agile mindset.

So instead of focusing on providing rigid and prescriptive rules for testing and planning out everything, we will focus on writing a guideline that are supportive example we will define what done means in the context of our project.

In terms of how we managed our code tests, we separated the unit and component tests into two separate testing projects in the GTL solution. We tried to regularly go over the tests we had written to verify whether these tests fulfill their objective and whether they are redundant or be improved in some way. During this project, we created, looked at, modified and deleted many tests. We did not document these changes using documentation, instead we used version control to track the updates we were making.

Test Control

Test control is a process of measuring progress and of tracking any changes in the testing plan, that could make us stray from our objectives. According to [2], "test control is an ongoing activity". We agree with this statement and because of the rapid development speed of agile, we found documenting test control tasks to be difficult. The only real place where the effects of test control are reflected is in the test plan. The way that we tracked progress was through using our Sprint board, and the test coverage metric.

When it comes to test traceability, we made sure that our tests and defects can be traced horizontally. We designed our tests such that they can be easily tracked back to their original test case.

Applied Test Techniques

In this section, we describe the test techniques and tools that we utilised during our testing activities.

Coding standards

We used coding standards as a tool for static testing. We followed the rules and best practices that we considered valuable, from a book called Clean Code [3]. One such example, was deciding beforehand on a naming convention for often used concepts:

- Creation of new objects = Create
- Adding existing objects = Add
- Changing already existing objects state = Update

- Deletion of objects = Delete
- Obtaining object information = Get

Code metrics

Code metrics are important to consider when writing quality code. In our case, we used Cyclomatic complexity, to identify the most complex parts of our system. A higher cyclomatic complexity usually indicates that the code is not very well written and that improvements should be seriously considered, so that refactoring of the code can be the next step.

Equivalence Class Partitioning

For the loan feature, we included the test cases that we obtained using a tool called equivalence class partitioning. The test cases are listed below:

Insert Loan Test Cases Equivalence Partitioning (EP)

Test Case No.	Valid test cases		
	ISBN	Barcode	CardNo
1	1234567891123	555555	1
	Invalid test cases		
2	1234556443455 (ISBN of book that does not exist)	11111	2
3	1234567891123	00000000 (barcode of copy that does not exist)	1
4	1234567891123	55555	0 (card number of cards that does not exist)
5	1230000000000 (ISBN of book with type that cannot be loaned)	44444	1
6	1234567891123	212121 (barcode of copy that is already on loan)	3
7	1234567891123	555555	01 (card no. belongs to expired card)
8	1234567891123	12121231 (barcode of book copy does not belong/ match to book (or item) with the specified ISBN)	5
9	12345621332222	333333	2 (holder of card no. has exceeded 5 book limit)

CheckBookType Unit Test

In this code example, we demonstrate we way that we implemented a unit test using NUnit. We tend to think of our unit tests as more white-box than black-box, because we are too familiar with the inner workings of the system at this point. In this case, the method under test is the CheckBookType function, located on the LoanController.

```
[Test]
[TestCase("Normal", true)]
[TestCase("Reference", false)]
[TestCase("Rare", false)]
[TestCase("Other", false)]
0 references | Maddie, 1 day ago | 1 author, 1 change
public void CheckBookType_Most_Cases_Return_Can_Loan(string typeOfBook, bool expectedResult)
{
    bool actualResult = loanCtr.CheckBookType(typeOfBook);

    Assert.That(actualResult, Is.EqualTo(expectedResult));
}
```

Figure 2: CheckBookType unit test code example

A neat feature of NUnit is that it allows us to define values for our test cases and then easily plug them into the unit test. Below, we added the business logic that is being tested by the code above.

```
1 reference | Maddie, 1 day ago | 1 author, 1 change
public bool CheckBookType(string typeOfBook)
{
    if (typeOfBook == "Normal") return true;
    else return false;
}
```

Figure 3: The implementation of the CheckBookType function

MatchBookIsbnwithBookCopyBarcode Integration Test

This integration test is used to test a function called suggestively MatchBookIsbnwithBookCopyBarcode. This function is used to test whether the book copy that a person wants to borrow is indeed a copy of the book that is being considered for borrowing. This function is located on the LoanController and it communicates with another component called LoanDAL that makes calls to the GTL database.

```
[Test]
[TestCase(1234567890, 77777777788, true)]
[TestCase(1234567890, 77777777780, false)]
0 references | Maddie, 1 day ago | 1 author, 1 change
public void MatchBookIsbnWithBookCopyBarcode(long isbn, long barcode, bool expectedResult)
{
    LoanController loanCtr = new LoanController();

    bool actualResult = loanCtr.MatchBookWithCopy(isbn, barcode);

    Assert.That(actualResult, Is.EqualTo(expectedResult));
}
```

Figure 4: MatchBookIsbnwithBookCopyBarcode integration test code example

Designing a testable architecture

There are several design principles and patterns that can be used to implement a more testable and maintainable architecture. Having an architecture that is easy to test, makes all the difference and justifies the effort put into refactoring or taking extra time to consider what patterns that could be used. For example, the Factory Method pattern is another variation of the well-known Factory design pattern. This pattern deals with object creation and it is based on the fact that the 'new' keyword should be avoided at all costs. The idea behind this pattern is that you are creating an object mould for the subclasses to choose from to instantiate. The Factory Method is a simple pattern to implement and could be considered a steppingstone before switching on to other more complex creational patterns such as the Abstract Factory pattern [4].

Another set of patterns that we used to make our architecture more testable are dependency injection and inversion of control. The principle of dependency injection basically states that high class objects should never depend on lower class and that we use inversion of control to reverse that control flow. A useful tool for implementing these patterns in the .NET Framework is the Unity Container tool.

Conclusion

We consider our solution for the GTL case the project where we tested the most and learned the most knowledge about testing in general, both theoretical and practical. We explored testing from an agile project point of view and we used various techniques and tools for both planning and implementing these tests.

References

[1] Lisa Crispin, Janet Gregory, “Agile Testing: A Practical Guide for Testers and Agile Teams”, Addison-Wesley Professional, 2009, ISBN: 9780321534460

[2] Black, Veenendaal, Graham, “Foundations of Software Testing ISTQ Certification”, Cengage Learning, 2012, ISBN: 9781408044056

[3] Robert C. Martin, “Clean Code: A Handbook of Agile Software Craftmanship”, Prentice Hall, 2008, ISBN: 9780132350884

[4] Factory Method Design Pattern, 2019, [Online]. Available: https://sourcemaking.com/design_patterns/factory_method