**Report of the Coursework ISAD157**

Used software:

* Microsoft Visual Studio C#(to work with application)
* MySQL Desktop application(to work with database)
* Draw.io (to draw diagrams)

Structure:

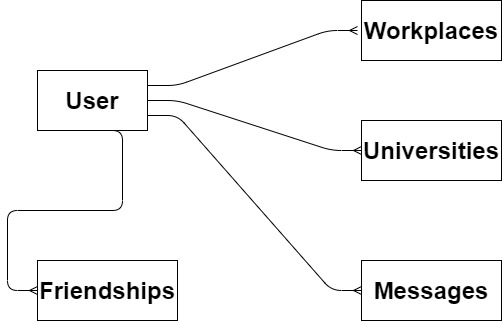
* Initial ERD
* Normalization
* Final ERD
* Database SQL
* Working with visual studio
* UML Diagrams
  + Use Case Diagram
  + Class Diagram
  + Activity Diagram
  + Sequence Diagram
  + State Diagram

**ERD**

What is a database? A database is a collection of structured information in a computer system.(Oracle) We organize a database in order to easily work with data. In our scenario we use relational type of databases, which means that our data will be organised as a set of tables, which have rows and columns, or records and attributes respectively. As we know, Entity-Relationship Modelling is a database description method that begins with determining entities(an object that contain the most significant data) and relationships between them( Connolly, T.M and Begg, C.E,(2005)).

At first, we need to identify our entities from the context of our work. So we have, User, friendships, workplaces, universities and messages main entities. Now, we need to define relations between this objects, from our given scenario we know that our user may have many friends, and he can also exchange messages with his/her friends. Therefore, we have one-to-many relationship between the users and messages, cause a user may have a different amount of messages. Also, from the Table 1 we can determine that a user may have a various number of universities and workplaces, that’s why another two one-to-many relationships between a user and workplaces and between a user and universities.

From the friends part of the table and the scenario, we can find out that the user can have many friends, but friend can also have many friends, and relation between a user and friends is many-to-many but we cannot use this type of relationships, hence we added friendships object. So we’ve got a one-to-many relationship between user and friendships. Now, we create an initial Entity-Relationship diagram.



**Normalization**

Now we need to load and store our data from a database in some order. Normalization is a process of designing a database, in which each table represents a single object and all attributes organised appropriately, according to the normalisation forms.(Connolly, T.M and Begg, C.E,(2005)). So we use four step normalization, we have Un-normalised Form, First Normal Form, Second Normal Form and Third Normal Form.

So, in un-normalised form we transfer all attributes from the Table 1, find an initial key and then define repeating groups. At first, we put all our attributes, they are User ID, First Name, Last Name, Gender, Hometown, City, Workplace name, University name, User ID2, Date time and Message. Our initial key will be User ID, because it is the only one attribute by which we can define a row (we cannot have users with equal user ID). Then we determined our repeating groups, as we know from our Entity-Relationship Diagram that a user may have many workplaces and universities we can affirm that attributes workplace name and university name is a repeating groups. The same situation will be with date time and messages, as a user may write many messages, in order to show our friendships entity we added an attribute User ID2, which also be a repeating group, according to the ERD we created .

UNF

User ID – initial key

First Name

Last Name

Gender

Hometown

City

(Workplace name) – repeating group

(University name)

(User ID2)

(Date time

Message)

After the un-normalised form we need to do first normalised form. Well, in first normalised form we separate our repeating groups, following which the appropriate key for each group is created and then propagate our initial key to the repeating groups. Well we have three repeating groups, for the workplace name our key will be composite key (two or more attributes needed for the identification) combination of User ID and workplace name, similarly with university name.

1NF

User ID

First Name

Last Name

Gender

Hometown

City

User ID

Workplace name

User ID

University name

User ID

User ID2

Date time

Message

According to the second normalised form, we need to separate any dependent attributes .Now we need to separate a friendships part and messages part. So we have a composite key User ID and User ID2 to define our friendships record and the same key will be for the messages.

2NF

User ID

First Name

Last Name

Gender

Hometown

City

User ID

Workplace name

User ID

University name

User ID

User ID2

User ID

User ID2

Date time

Message

Finally, the third normalised form, we need to define all keys including foreign key(the non-key attribute that is the key of other relation).

3NF

User ID

First Name

Last Name

USER

Gender

Hometown

City

User ID

Workplaces

Workplace name

User ID

University name

Universities

User ID\*1

Friendships

User ID2

User ID\*2

User ID2

Date time

Messages

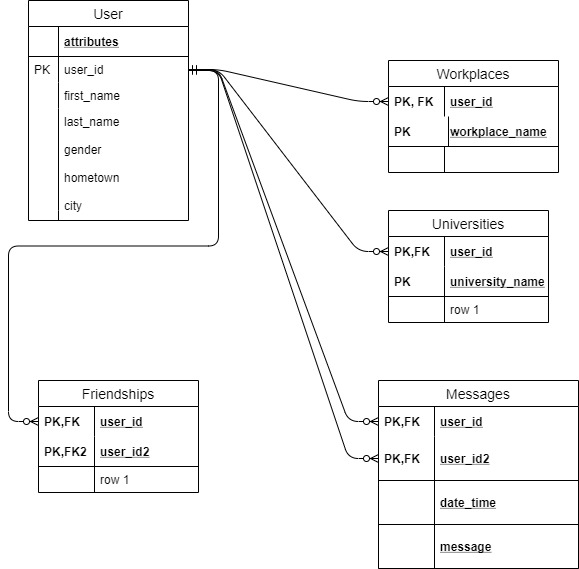
Message

\*1 means that in this table we have a composite key User ID + User ID2

\*2 means that in this table we have a composite key User ID + User ID2

User ID will be the foreign key for the universities, workplaces, friendships, and messages.

**Final ERD**

Now we need to combine our normalisation results and initial Entity-Relationship diagram.

**Database SQL(creating tables and relationships)**

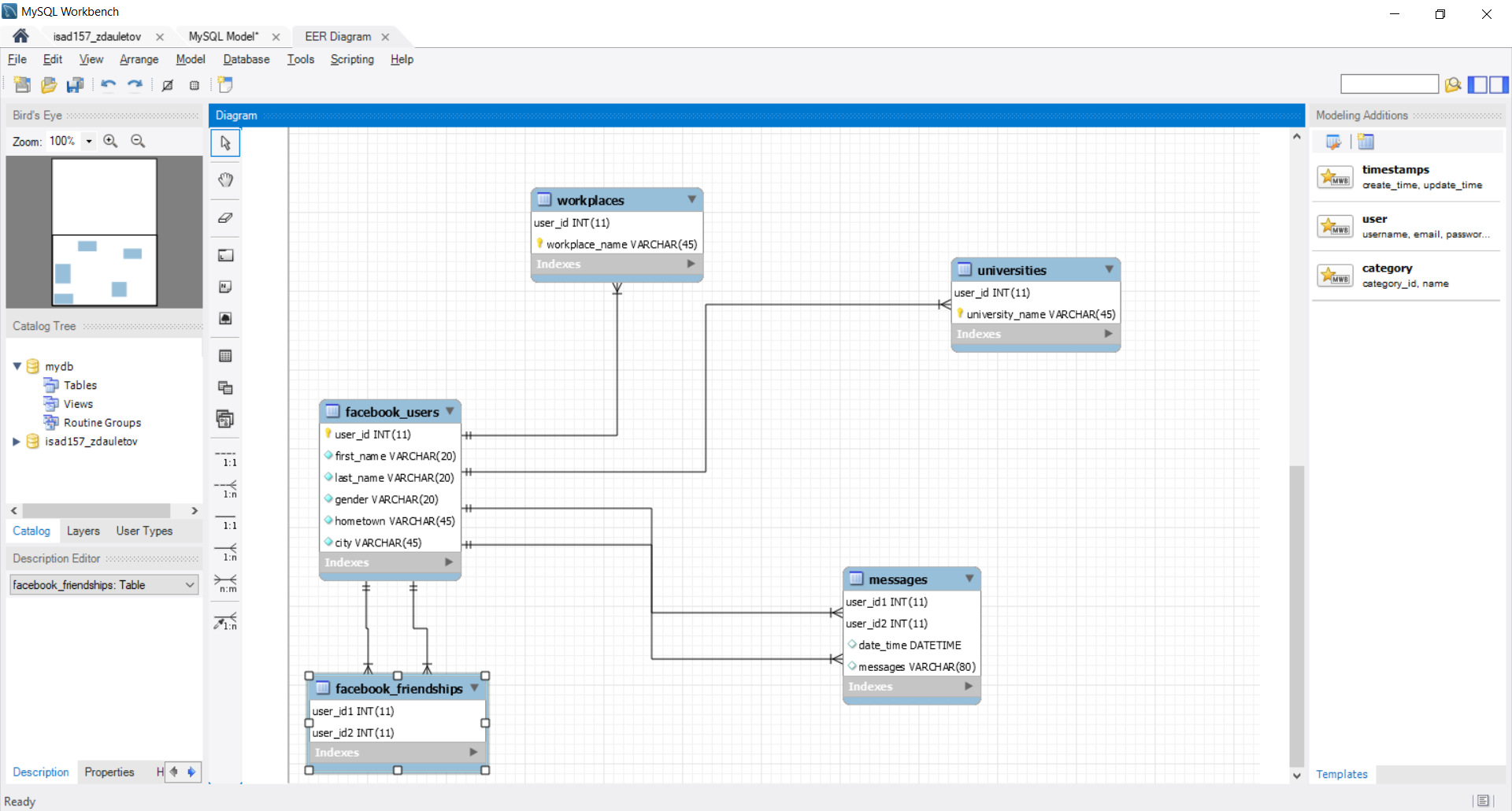
After we did final Entity Relationship diagram, we can implement to Database Management System(MYSQL), which is working on Structured-Query Language. According to the final ERD diagram we have five entities, consequently we have five tables with the same names. So in order to create a database table we use the following syntax(W3Schools, Create Table):

CREATE TABLE table\_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
   PRIMARY\_KEY(column 1),  
);

To add a foreign key we use ALTER TABLE syntax(W3Schools, Foreign Key):

ALTER TABLE table\_name1  
ADD FOREIGN KEY column\_name1 REFERENCES table\_name2.column2;

So, we have facebook\_users, facebook\_friendships, universities, workplaces, messages tables and we have attributes for them, which will be our columns. When we create a foreign key, when we automatically create a relationship between tables or entities. Now, we need to illustrate how our database objects are connected, in order to do it we use reverse engineer, which will create model similar with our final Entity-Relationship Diagram. Thereafter, we loaded our csv files into the tables.



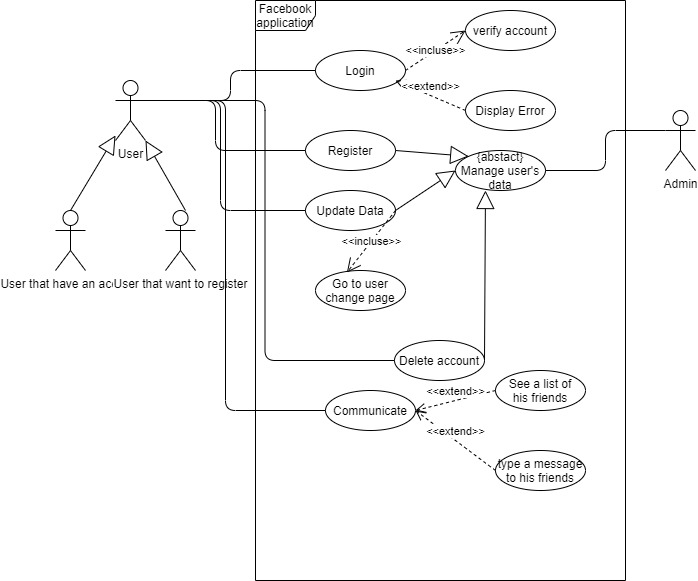
**UML Diagrams**

The Unified Modelling Language(UML) is technique that widely represents aspects of software system, it is very useful for company to have UML diagrams, because it simplifies communication between a customers and developers. As we know that misunderstanding can cost to company an enormous amount of money(Martina Seidl et al. 2015 and UML@ Classroom, 1-2 pages).Well, in our application we use the following diagrams: use case diagram, class diagram, activity diagram, sequence diagram and state diagram.

**Use Case Diagram**

In order to represent the software system from user’s point view, the use case diagram is used. In this diagram developers or managers only show the functionalities, without demonstrating implementation details of the system(Martina Seidl et al. 2015 and UML@ Classroom, 1-2 pages 46-47). The main parts of this model are system, use case, actor, generalization and relationships. The system in our case will be facebook application, which contain a database of users. A use case shows actions with the system, that a customer can do with system, in our scenario a user can login or register to the database, update his personal information, delete his account and communicate with the other users of the application.

An actor is someone who interacts with the application, in our case he can login or register, update his data and etc. From given scenario, we pick out two main actors, a user and an admin(developer) who maintain the system, an admin is added, because if a user does not comply with user requirements, for instance he insults the other user, an admin can delete his account from the system. In addition, we added two secondary characters, a user who have an account, and those who wants to register. A user will inherit from secondary actors, this means that relationship between them will be generalization. Also we have an association, in our diagram a user connected with the use case login via association which express that the actor must communicate with the system.

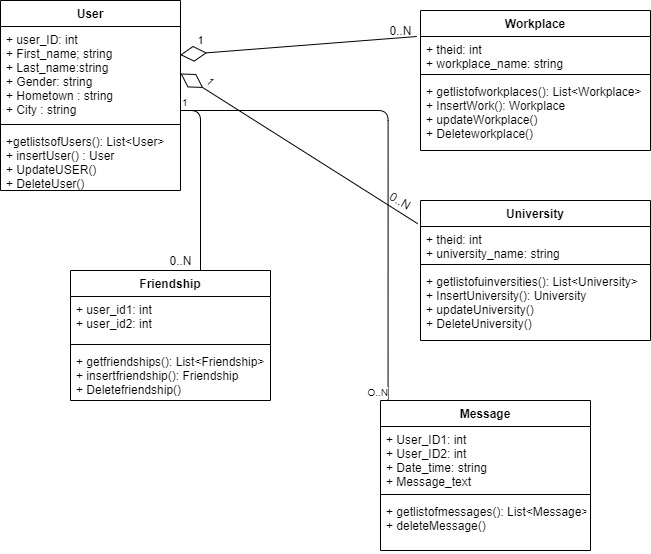
When a customer tries to login to the system, he puts his personal information and then system have two cases, customer puts a correct data or not, if he did everything right he will get a verify account use case with a relationship include, as it happens on a constant basis. And other user case display error, it will be an extend relationship between a login and display error, as it is not a permanent case. From the chart below and statements above, it has been noticed that we have a classic use cases and a secondary use cases(verify account, display and et.) which means that they performs after the major functionalities(login, register, and et.al).Also we notice an abstract use case it generalizes three use cases and it is also a functionality of the admin. 

**Class diagram**

After we have done a use case diagram, we understand how the customer communicates with the system, now we need to put an emphasis on structure of the system, how the attributes are connected. Class diagram describes the units of the application and how they relate to each other, these elements are permanent. (Martina Seidl et al. 2015 and UML@ Classroom, 49 page)A UML class diagram consists of two parts: classes and relationships between them.

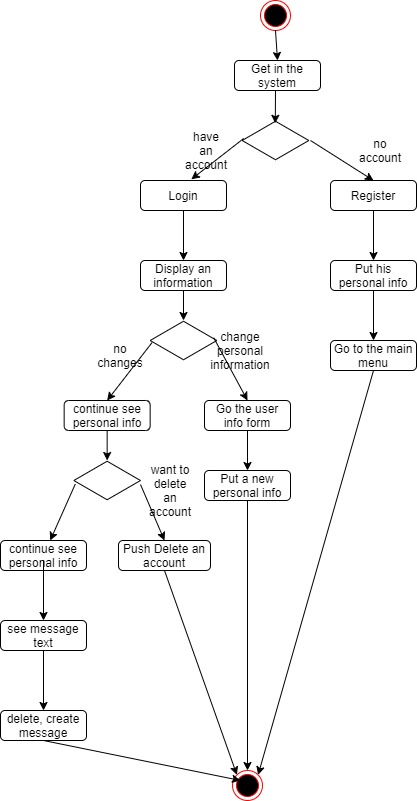
A class is description of a set of objects, class consists of three parts: class name, class attributes and class methods or functions. In our case we can pick out five classes similar with tables in SQL: user, friendship, workplace, university and message. Attributes in them will be similar with Final Entity-Relationship Diagram. Now, we need to determine our operations(methods). At first we need to display some table in our application functions with beginning get will send a list (collection of data, like an array) of attributes which will be accessed from the MySQL database. Then we need to add some record to our database it will be operation beginning with insert, and also we need change our data, an update functions will help us with idea, and finally we need to delete a data we use functions delete, and then we put our functions to all classes, exception we cannot update our friendship, as it is impossible, also we cannot update a message.

Now, we need to connect our classes, that’s why we use relationship. They are: accosiation, inheritance and aggregation. An aggregation means that one class(child class) cannot exist without the other class(parent class), in our case we have two aggregations, between workplace and user, between university and user, a user will be a “parent” class. In our scenario, we also have an association it is friendship and message with a user, the same case with Use Case Diagram.

****

**Activity diagram**

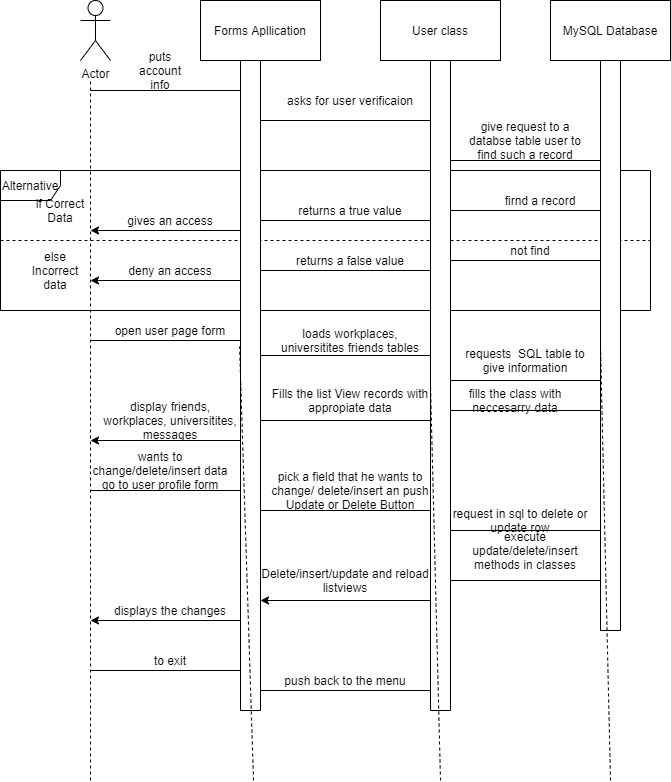
An activity diagram is a chart, which describe our application activities, in our scenario, we can login to our system in an entry form, activities is an action, which user can execute, activities are inside the rounded rectangle. Also we use if cases, because we have several possibilities. For example, a user have two cases login or register, whether he has or has not an account. In activity diagram, we describe a sequence of activities. Diamonds are our conditions and circles are beginning and ending.

****

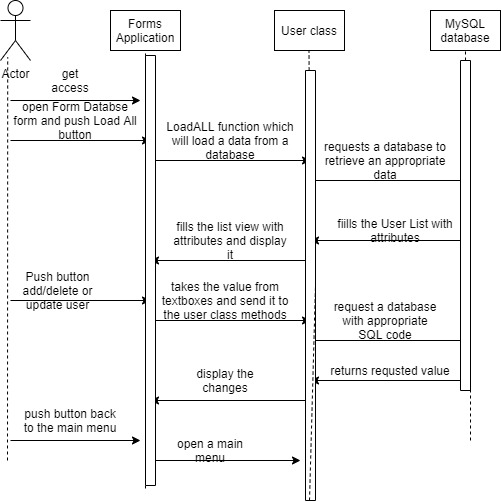
**Sequence Diagram**

A sequence diagram describes objects interactions(Wilipedia), in our case from Use Case diagram, class diagram and activity diagram, we highlight three objects our forms application, User class(contains all classes) and also we have an abstract class a database. Sequence diagram show activities sequent. In this diagram we also have actors, that’s why we need two diagrams for user and for admin.

For user:



For admin:

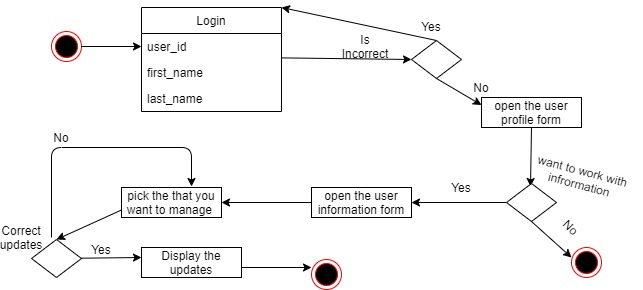


**State Diagram**

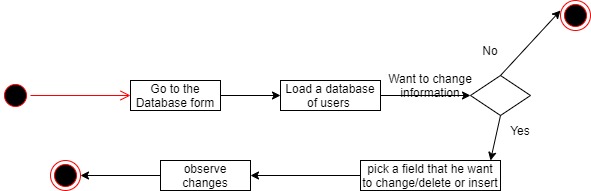
Finally, the last UML diagram in this report is a state diagram, which is used to illustrate the states in which a system or an object can find itself during its live, the time between creation and destruction. In this diagram we also have activities or events, and we also have condition like in activity diagram and sequence diagram.

When a user open a facebook application, he see a form with three fields, user id, first name and last name, we cannot exclude user id, cause the system may have several John Smith, in order to separate them we need a unique identification number, as we know from Entity-Relationship Diagram, a user\_id is our initial key. In addition, like in sequence diagram we have two charts, for user and for admin.

For user:

****

For admin:

****

**Conclusion**

Taking all diagrams and SQL code above we can start our application in visual studio. We will have several form entry, form register, form user page, form user information and form database for admin, also we add a class user, in which we put all our classes and their methods. So we in user page we have list view for friends and in form user information we have two list views with friends and messages.

**Reference List**

(Oracle, Database, Oracle, viewed 25 April 2020, <https://www.oracle.com/database/what-is-database.html>)

(Connolly, T.M. and Begg, C.E.,(2005). Database systems: A practical approach to design, implementation, and management, Person Education, 342-343 pages)

(Connolly, T.M. and Begg, C.E.,(2005). Database systems: A practical approach to design, implementation, and management, Person Education, 387-389 pages)

(W3Schools.com, SQL Create Table Statement, viewed 25 April, <https://www.w3schools.com/sql/sql_create_table.asp>)

(W3Schools.com, SQL Foreign Key Constraint, viewed 25 April, <https://www.w3schools.com/sql/sql_foreignkey.asp>)

(Martina Seidl, Marion Scholz, Christian Hummer, Gerti Kappel (2015), UML@ Classroom, An Introduction to Object-Oriented Modeling, Springer)

(Wikipedia, Sequence diagram, viewed 25 April, <https://en.wikipedia.org/wiki/Sequence_diagram>)