Introduction

Within this coursework I will be setting out a plan for a piece of software of my chosen scenario, based on the data given; using the appropriate UML models as well as a series of entity relationship diagrams, both initial and normalised. I will also be coding a basic user interface in order to display a large data set appropriately. All diagrams will be drawn in draw.IO as well as described/annotated.

My chosen scenario will be a design for the police department, to be used to scope out suspected criminal activity by looking into the suspected perps social media message and friend connections.

GIThub Link

<https://github.com/La-Ola/ISAD157>

Requirements

Functional requirement for this application include:

* Storing users names
* Storing user IDs
* Storing users friends
* Store messages
* Store workplace
* Store school
* Allow users to interact with data- interface

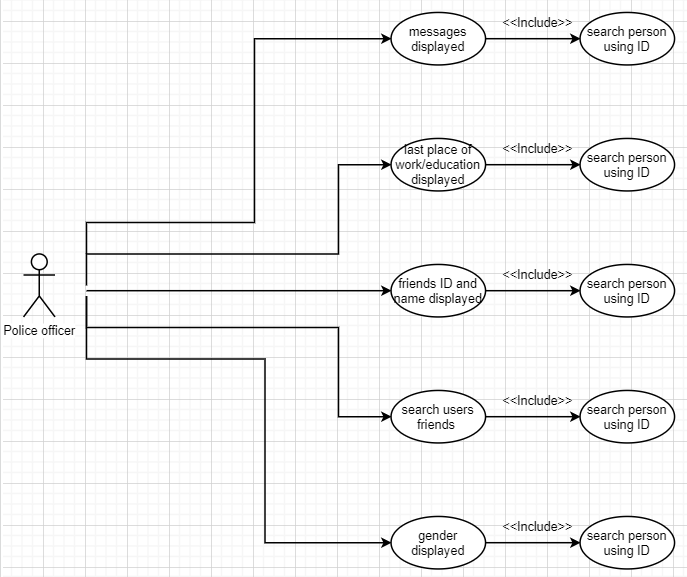
Non-functional requirements:

* Legal
  + Security using a hashing table
  + GDPR – consent to store data
* Ethical
  + Storing peoples data long term
  + Viewing people data
* Usability; interface must conform to user accessibility rules

User stories

* As a police officer I want to enter a userID number and be displayed all their messages so that I can look for potentially incriminating evidence of a perp.
* As a police officer I want to enter a userID and be shown their last place of work/education so that I can follow leads.
* As a police officer I want to enter a userID and be shown their friends ID and names so that I can see if they are friends with other perps.
* As a police officer I want to be able to search through a user’s friends so that I can quickly see if there are associations between perps.
* As a police officer I want to enter a userID and be shown the gender of the perp so that I can match witness statements.

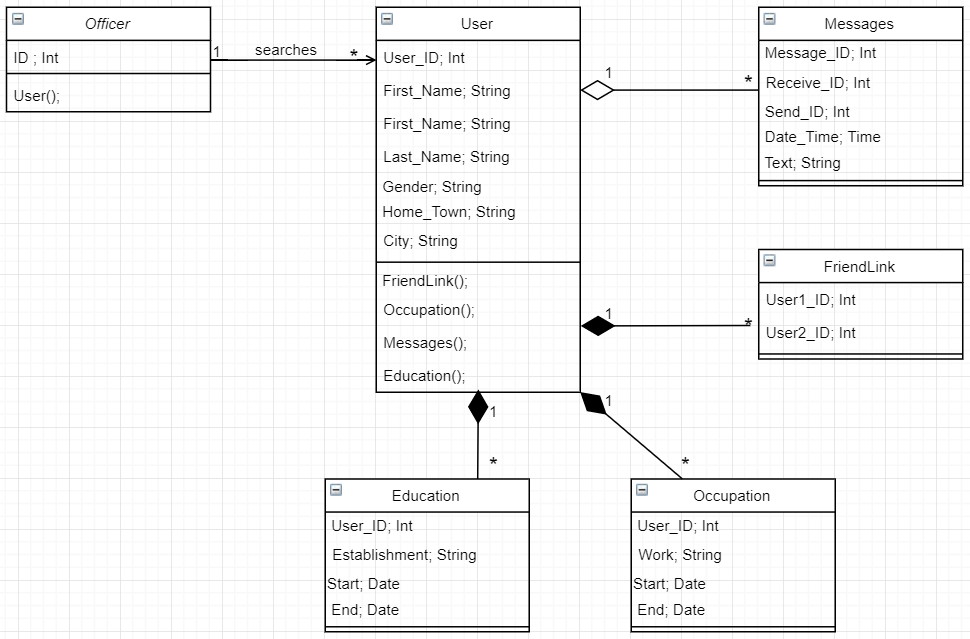
Use Case Diagrams



(Flowchart Maker & Online Diagram Software, 2020)

This use case diagram easily displays the different aspects of the software that the actor can explore; whilst showing the client what the developer grasps as most important functions to include.

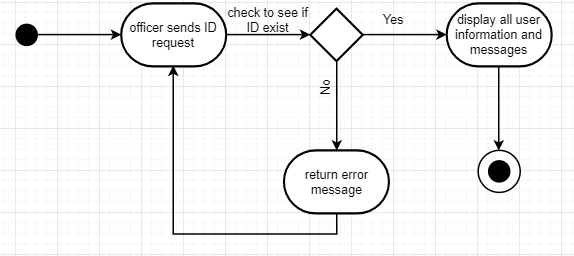
Class Diagram



(Flowchart Maker & Online Diagram Software, 2020)

The class diagram is used to represent the static view of an application. It’s a strong and effective way to structure classes so that they can be translated in to an object orientated languages. It’s also a brilliant way to show the client how their data links together, giving them a higher understanding of their software. It uses both aggregation and composition arrows to convey the dependency on the main class ‘user’.

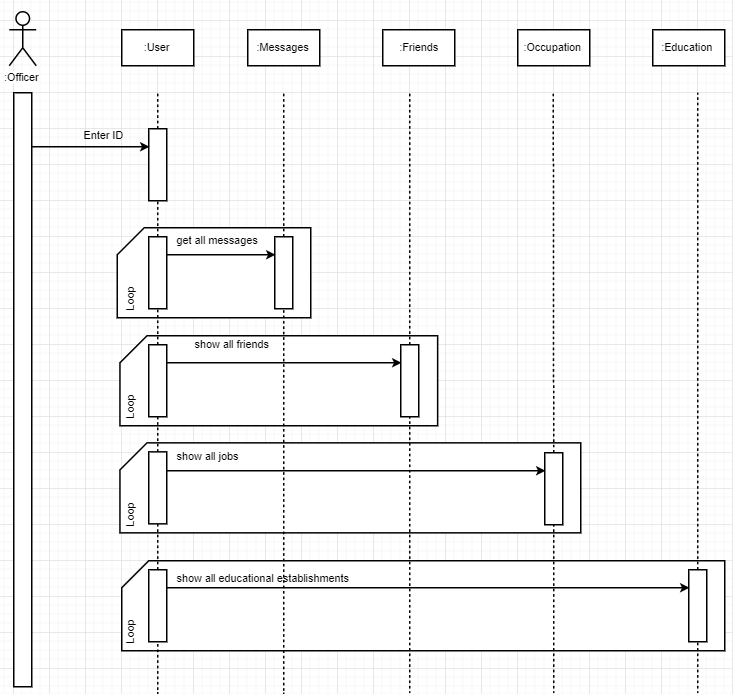
Activity Diagram of information retrieval



(Flowchart Maker & Online Diagram Software, 2020)

I have used an activity diagram to display the flow of information through the application to be made. The diagram moves from the start terminal to the first an action, which is an input from the user. Is then runs through a condition check, allowing the system to come to a termination.

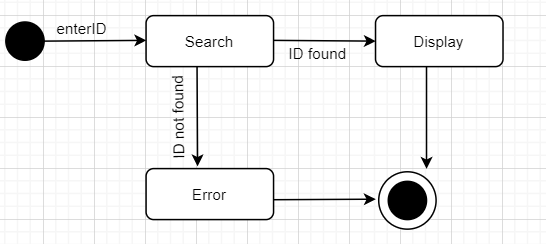
Sequence diagram



(Flowchart Maker & Online Diagram Software, 2020)

A sequence diagram is key to laying out the order of operations within a system. As the diagram descends the time of program execution is increasing; therefore, the first command will be made by the actor, Officer, by inputting the ID of the user they wish to search. After doing this, 4 loops are executed to claim information about the specific user.

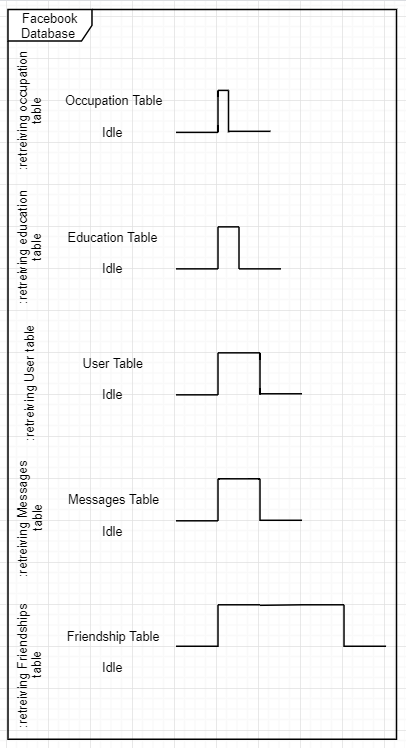
State diagram



(Flowchart Maker & Online Diagram Software, 2020)

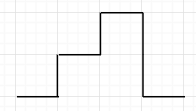
The state diagram is used to emphasise the states in which objects pass through. It is also used to match the systems states to real life states. The comments on each arrow indicate actions, they are used as transition commands between states.

Time Diagram

A time diagram is much like a sequence diagram in that its designed for some form of time monitoring.

The length of the peaks determine the length of time it requires to perform the action specified; for instance, the time it takes to retrieve the occupation table is about half that of the time it takes to retrieve the education table, which is again about half the time it requires to retrieve the user table.

It should be noted that this is a simple time diagram, only displaying the retrieval of data, each object only has one function; if I were to add to the object, displaying the data, the diagrams would look more like this…



(Flowchart Maker & Online Diagram Software, 2020)

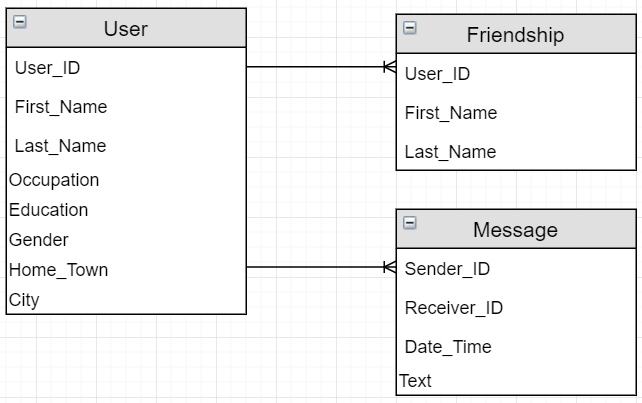
(Flowchart Maker & Online Diagram Software, 2020)

UNF and Corresponding Initial ERD

User

User\_ID

First\_Name

 Last\_Name

Occupation

Education

Gender

Home\_Town

City

Friendships

User\_ID

First\_Name

Last\_Name

Messages

Sender\_ID

Receiver\_ID

Date/Time

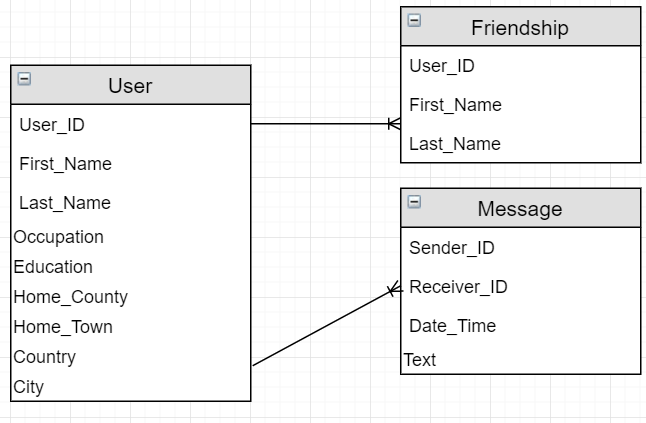
Text

(Flowchart Maker & Online Diagram Software, 2020)

The un-normalised form shows the initial structure of the table, and the ERD shows how they would connect together. Although all relationships are one to many, this ERD and UNF are both in a juvenile state and have a long transformation before they are fully functioning. At this stage, the information laid out is based on the initial table in the brief, table 1.

1NF

User

 User\_ID

First\_Name

Last\_Name

Gender

Home\_Town

Home\_County

City

Country

Friendships

User\_ID

First\_Name

Last\_Name

Messages

Sender\_ID

Receiver\_ID

Date/Time

(Flowchart Maker & Online Diagram Software, 2020)

Text

The 1st normalisation stage is the simplest excepted form to build a table, it isn’t efficient nor is it well designed, but it is a key stage for further normalisation. In the 1st normalisation, all comma separated values have been split into their own columns; this has been done for hometown and city; within table 1, hometown is separated with a comma between the town and county, so to resolve this, a county column must be created. Same principle for the city; the country is included with a comma, therefore a new column must be made.

2NF

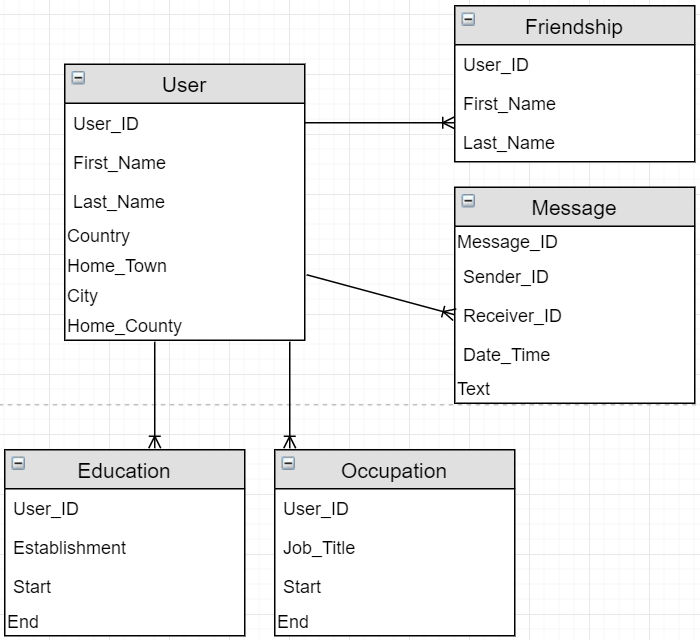
User

User\_ID

First\_Name

Last\_Name

Gender

 Home\_Town

Home\_County

City

Country

Friendships

User\_ID

First\_Name

Last\_Name

Messages

Message\_ID

Sender\_ID

Receiver\_ID

Date/Time

Text

Occupation

User\_ID

Workplace

Start

Terminate

Education

User\_ID

Establishment

Start

(Flowchart Maker & Online Diagram Software, 2020)

Graduation

During the second normalisation, it is obvious to see that two new flat files have been added to form our relational database. During the first normalisation, it is key to separate out columns that can contain multiple pieces of data into their own table; this has been done with the education and occupation column because each user could have had more than one job or more than one educational establishment that they have attended. Appropriate columns have been created and named for these new tables. The key from its original table, user, gets propagated into the new tables created, this then becomes a foreign key for the new tables. As well as this, the composite key within ‘Message’ has been resolved by adding a unique identifier for each row.

3NF

User

User\_ID

First\_Name

Last\_Name

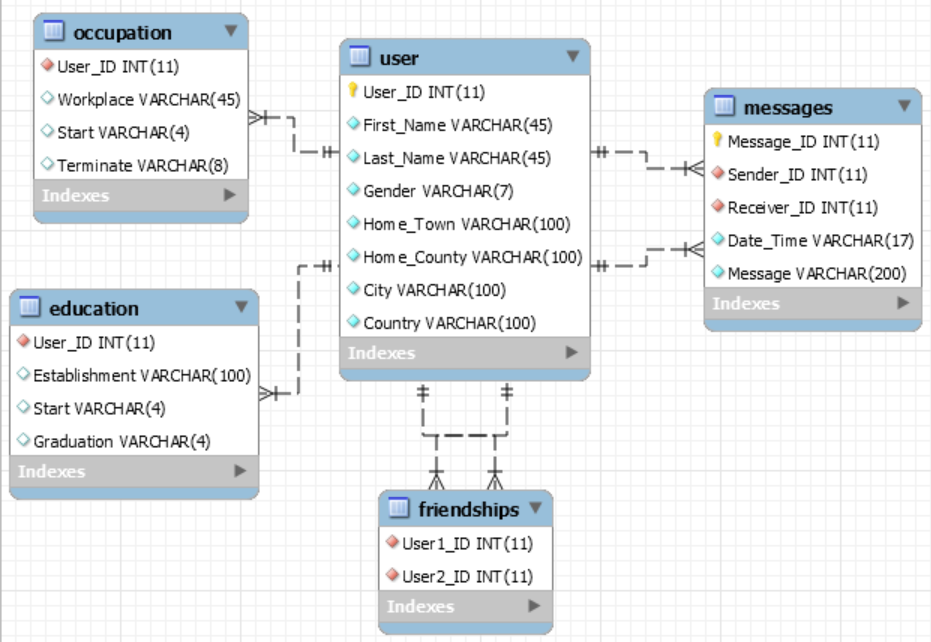
Gender

Home\_Town

Home\_County

City

Country

Friendships

User1\_ID

User2\_ID

Messages

Message\_ID

Sender\_ID

Receiver\_ID

Date/Time

Text

Occupation

User\_ID

Workplace

Start

Terminate

Education

User\_ID

Establishment

Start

(Workbench)

Graduation

The 3NF diagram was generated in workbench. It is directly related to the SQL tables and contents. The friendship table has been refactored so that less data needs changing is a document change occurs. Foreign keys are indicated by the red rhombi, and primary keys by the golden key. Both messages and friendships table have multiple foreign keys therefore have multiple one to many connections.

SQL statements

CREATE TABLE isad157\_sskinner.user (

User\_ID INT NOT NULL,

First\_Name VARCHAR(45) NOT NULL,

Last\_Name VARCHAR(45) NOT NULL,

Gender VARCHAR(7) NOT NULL,

Home\_Town VARCHAR(100) NOT NULL,

Home\_County VARCHAR(100) NOT NULL,

City VARCHAR(100) NOT NULL,

Country VARCHAR(100) NOT NULL,

PRIMARY KEY (User\_ID));

CREATE TABLE isad157\_sskinner.friendships

(

User1\_ID INT NOT NULL,

User2\_ID INT NOT NULL);

CREATE TABLE isad157\_sskinner.messages (

Message\_ID INT NOT NULL AUTO\_INCREMENT,

Sender\_ID INT NOT NULL,

Receiver\_ID INT NOT NULL,

Date\_Time VARCHAR(17) NOT NULL,

Message VARCHAR(200) NOT NULL,

PRIMARY KEY (Message\_ID),

UNIQUE INDEX idmessages\_UNIQUE (Message\_ID ASC) VISIBLE);

CREATE TABLE isad157\_sskinner.occupation (

User\_ID INT NOT NULL,

Workplace VARCHAR(45) NULL,

Start VARCHAR(4) NULL,

Terminate VARCHAR(8) NULL);

CREATE TABLE isad157\_sskinner.education (

User\_ID INT NOT NULL,

Establishment VARCHAR(100) NULL,

Start VARCHAR(4) NULL,

Graduation VARCHAR(4) NULL);

All of the above are SQL statements. Each section is used to create a table within the database isad157\_sskinner, based on the server held at Plymouth university. These statement were executed on MySQL workbench. Within these statements, primary keys are specified in each table. Data types have been set with limits and whether or not the data slot is allowed to be empty or if it must be filled, this is determined by “NULL” and “NOT NULL” respectively

ALTER TABLE isad157\_sskinner.education

ADD INDEX User\_ID\_idx (User\_ID ASC) VISIBLE;

;

ALTER TABLE isad157\_sskinner.education

ADD CONSTRAINT User\_ID

FOREIGN KEY (User\_ID)

REFERENCES isad157\_sskinner.user (User\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION;

ALTER TABLE isad157\_sskinner.occupation

ADD CONSTRAINT Userr\_ID

FOREIGN KEY (User\_ID)

REFERENCES isad157\_sskinner.user (User\_ID

ON DELETE NO ACTION

ON UPDATE NO ACTION;

ALTER TABLE isad157\_sskinner.friendships

ADD INDEX Useerr\_ID\_idx (User1\_ID ASC) VISIBLE,

ADD INDEX Useer\_ID\_idx (User2\_ID ASC) VISIBLE;

;

ALTER TABLE isad157\_sskinner.friendships

ADD CONSTRAINT Useerr\_ID

FOREIGN KEY (User1\_ID)

REFERENCES isad157\_sskinner.user (User\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

ADD CONSTRAINT Useer\_ID

FOREIGN KEY (User2\_ID)

REFERENCES isad157\_sskinner.user (User\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION;

ALTER TABLE isad157\_sskinner.messages

ADD INDEX Uuser\_ID\_idx (Sender\_ID ASC) VISIBLE,

ADD INDEX Usser\_ID\_idx (Receiver\_ID ASC) VISIBLE;

;

ALTER TABLE isad157\_sskinner.messages

ADD CONSTRAINT Uuser\_ID

FOREIGN KEY (Sender\_ID)

REFERENCES isad157\_sskinner.user (User\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

ADD CONSTRAINT Usser\_ID

FOREIGN KEY (Receiver\_ID)

REFERENCES isad157\_sskinner.user (User\_ID)

ON DELETE NO ACTION

ON UPDATE NO ACTION;

on the page above is the SQL code for the addition of foreign keys to each table, allowing them to be interlinked, and have data accessed just by having access to the user table. Constraint names must all be individual for these, to do this I have added a letter somewhere different in “User\_ID” for each foreign key.

Conclusion

The more UML diagrams used the easier it is to portray your vision of a piece of software to a client, ensuring they are getting exactly what they want out of your services. Normalisation diagrams are key to ensuring that all data redundancies and composite keys are removed in a logical fashion and the most optimised version of your database can be created.

References

Draw.IO. 2020. *Flowchart Maker & Online Diagram Software*. [online] Available at: <https://app.diagrams.net/> [Accessed 20 March 2020].