ISAD157 – Facebook Database Coursework Report

*Introduction:*  
For this coursework project, it was decided that the database would be built for an investigator working for law enforcement. This investigator would have basic access to the database, being able to view any of the necessary information but not edit it in any way. The scenario in mind was that the investigator would have been given access to a particular sample in a third-party’s database that had been identified within the entire database for potentially suspicious users and/or messages – finding this information was a part of the case to which the investigator had been assigned. The investigator was to analyse the users, their information, and their messages to find anything that would aid their investigation. The second (and only other) user for this database (sample) is the third-party’s database manager themselves. This person would have higher-level access than the investigator; they would have the additional capability to create, read the information about, update the information about, and delete users. Though the manager’s capabilities were not represented within the database itself, they were shown through the UML diagrams.

*Evaluation:***UML Diagrams:**The first diagram to be created (and the first to be finished) was the Use Case Diagram. It is simple enough, containing two separate diagrams: one for each user (the ‘Social Media Investigator’ and ‘Third-Party Database Manager’, abbreviated to SMI and TPDM respectively for conciseness and simplicity).   
[20.04.02 Facebook Use Cases (Coursework)](20.04.02%20Facebook%20Use%20Cases%20(Coursework))

**Entity Relations:**

**SQL Query Statements:**The Users table was one of the simplest to create, containing no foreign keys and only data of type VARCHAR (variable characters). It had one primary key, the “user\_id”, and was the basis for the connection between the other tables. It had no need to be edited at any point. The VARCHARs were set to limits that were thought to be appropriate – their values encompassed what was thought to be the likely maximum limit for each attribute (for example, double-barrelled surnames and those with two forenames were two considerations that affected the decision about the values used). It should be noted that the “relationship\_status” attribute was dropped completely; although it can be found within the normalisation, it was not included with the given dataset, and the importing failed when the column was present. As a result, it was removed with little intention on bringing it back because of the lack of data that would accompany it. It would be essentially adding an empty column and using it for nothing.  
CREATE TABLE isad157\_jkinver.users  
(  
user\_id VARCHAR(15) NOT NULL,  
user\_forename VARCHAR(40) NOT NULL,  
user\_surname VARCHAR(40) NOT NULL,  
user\_hometown VARCHAR(50) NOT NULL,  
user\_gender VARCHAR(12) NOT NULL,  
relationship\_status VARCHAR(20) NOT NULL,  
user\_town\_or\_city VARCHAR(50) NOT NULL,  
PRIMARY KEY (user\_id)  
);

The Workplace(s) table was the subsequent table created. It wasn’t quite as simple as the Users table as it contained a composite key and a foreign key, both of which were manually programmed in through query statements (rather than by use of the wizard). It may have been the case that the ‘constraint’ was unnecessary, but it was seen fit as to better the understanding of coding a foreign key (the understanding being that the foreign key itself was the “user\_id”, where its ‘name’, if you will, was “workplace\_user\_id”. It should be noted that, similar to “relationship\_status” from the Users table, the “user\_workplace\_date” was dropped completely from the Workplace(s) table. Again, no data was given for the column, rendering it unnecessary as far as the database was concerned, despite its inclusion in the normalisation.  
CREATE TABLE isad157\_jkinver.workplaces  
(  
user\_id VARCHAR(15) NOT NULL,  
user\_workplace VARCHAR(60) NOT NULL,  
user\_workplace\_date DATE NOT NULL,  
PRIMARY KEY (user\_id, user\_workplace),  
CONSTRAINT workplace\_user\_id FOREIGN KEY (user\_id)  
REFERENCES isad157\_jkinver.users(user\_id)  
ON UPDATE CASCADE  
ON DELETE CASCADE  
);

The Schools/Universities table was virtually identical to the workplaces table. It too included two primary keys forming a composite key, a foreign key, and the same datatypes (even down to the number of characters for the VARCHARs). This was deliberate, as its function was almost exactly the same: the only difference between the two really was the content for where one table detailed the workplace(s) of each user, the other stated the educational establishments of which they were a part. Such repetition is also arguably a reasonable design choice: it demonstrates consistency in the database, which might arguably reinforce the user’s understanding of the UI. Another similarity between this table and the Workplace(s) (as well as the Users) table was the exclusion of a field. And, as was the case for the Workplace(s) table, the date column was the one to be dropped for the same reason as the other two: for the normalisation, it is a necessary inclusion, but for the database, it can be afforded to be ridden of.   
CREATE TABLE isad157\_jkinver.schools  
(  
user\_id VARCHAR(15) NOT NULL,  
user\_school\_or\_uni VARCHAR(60) NOT NULL,  
user\_school\_or\_uni\_date DATE NOT NULL,  
PRIMARY KEY (user\_id, user\_school\_or\_uni),  
CONSTRAINT school\_user\_id FOREIGN KEY (user\_id)  
REFERENCES isad157\_jkinver.users (user\_id)  
ON UPDATE CASCADE   
ON DELETE CASCADE  
);

The User-Friend Link table was arguably the simplest table to create. It only had two fields, both of which formed a composite key and one of which was a foreign key, and the only datatype used for it was, again, VARCHAR. Though initially it was thought of as a good idea to make both fields foreign keys so as to link the now-removed Friends table to the others, it was realised (with the removal of the Friends table) that only the user\_id needed to be a foreign key, as that would be enough to link the User-Friend Link table to the other tables. The former Friends table had no link otherwise, but with the dataset not actually containing a separate file for friends’ IDs, forenames, and surnames (on account of the significant point that friends are themselves considered users for this database), it was arguably an unnecessary inclusion, despite its place in the normalisation. Hence, the table was dropped. This is also the reason for the somewhat strange naming of this table compared to the others.   
CREATE TABLE isad157\_jkinver.user\_friend\_link  
(  
user\_id VARCHAR(15) NOT NULL,  
friend\_id VARCHAR(15) NOT NULL,  
PRIMARY KEY (user\_id, friend\_id),  
CONSTRAINT link\_user\_id FOREIGN KEY (user\_id)  
REFERENCES isad157\_jkinver.users (user\_id)  
ON UPDATE CASCADE   
ON DELETE CASCADE   
);

Again, the Friends table was seen, with regards to the database, as an unnecessary inclusion as it would only cause the repeat of data, which is something to avoid in a relational database. Following is the code used to create the Friends table before it was completely removed:  
CREATE TABLE isad157\_jkinver.friends  
(  
friend\_id VARCHAR(15) NOT NULL,  
friend\_forename VARCHAR(40) NOT NULL,  
friend\_surname VARCHAR(40) NOT NULL,  
PRIMARY KEY (friend\_id)  
);

The Messages table was another table that was, initially, not overly complex, containing only a composite key and a total of four fields. There lay a problem, however, which was that it ended up not having a direct relation to any other table in the database (which was seen in the first Final ERD). Resultantly, the decision to make a foreign key to relate them was made. It was additionally realised that the “sender\_id” and “receiver\_id” were interchangeable to some extent, and with friends themselves being users, it was seen as fit to relate both values to the “user\_id”; both the sender and receiver were of course their own user, after all. Because of this, two foreign keys were made for this table, and because the thought materialised after the creation *and* population of the table, it was not a simple option to drop and re-code the table in this case: the wizard was admittedly used to create these two particular foreign keys. As such, the code that follows does not contain the creation of the two foreign keys (though above query statements prove it is known how to create them manually).  
CREATE TABLE isad157\_jkinver.messages  
(  
sender\_id VARCHAR(15) NOT NULL,  
receiver\_id VARCHAR(15) NOT NULL,  
date\_and\_time\_sent DATETIME NOT NULL,  
content VARCHAR(640) NOT NULL,  
PRIMARY KEY (sender\_id, receiver\_id)  
);

The only other query statements to show are the SELECT statements used by MySQL through C#. The queries were declared as strings each named “query” in the C# program and their values were assigned to exactly what is shown below (one line per query). Through the use of these strings, the tables in the database could be viewed through the C# application in data grids: one grid for each table.  
"SELECT \* FROM isad157\_jkinver.users"  
"SELECT \* FROM isad157\_jkinver.user\_friend\_link"  
"SELECT \* FROM isad157\_jkinver.workplaces"  
"SELECT \* FROM isad157\_jkinver.schools"  
"SELECT \* FROM isad157\_jkinver.messages"

*Conclusion:*