ADVANCED DATABASE SYSTEMS

PROJECT

MAY 2018

Sydney Twigg
M8C3XRSN8

CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

QUESTION 1

TechDating - an online dating application.

My chosen scenario is to create a dating application wherein users can register and be matched with other users and exchange messages with nearby users who meet their criteria. The project will be created using Java and SQL and will be designed and developed using NetBeans and MySQL Workbench. A fully functional database that cohesively integrates with the application should be produced.

The specifications for a fully functional database have been outlined as the following:

- ERD modelling
- Data normalization - 3rd level
- Entity integrity maintained
- Referential integrity maintained
- Cohesive modelling of a conceptual model to a logical model
- Entity types identified and understood
- Relevant indexes identified and implemented
- Relevant views identified and implemented
- Cohesive integration between application or website and database
- Database runs

The requirements for the database are as follows:

- 5 entities
- 1 trigger
- 2 stored procedures
- 2 indices
- 2 users

All of the above requirements should be met in order to produce a fully functional database system for the TechDating application.

The TechDating System specifications:

Identify the entities to be used in the system:

- Member
- Login
- Match
- RelationshipInterest
- Message
- Gender
- Orientation

The following table outlines what each entity will be used for in the TechDating System:

Entity	Description
Member	Table to store all member's personal details to be displayed on their profile.
Login	Table to store all member's login details in order for them to login to the system.
Match	Table to the member ID's of matched members.
RelationshipInterest	Table to store a list of accepted relationship interest inputs.
Message	Table to store all messages sent between two members.
Gender	Table to store a list of accepted gender inputs.
Orientation	Table to store a list of accepted sexual orientation inputs.

Table 1 - Names and description of each entity in the TechDating System.

The following relationships will exist between the entities:

- Member and Login will have a one-to-one relationship, where each member has their own login details.
- Member and Match will have a one-to-many relationship, where each member can have many matches.
- Member and Gender will have a one-to-many relationship, where many members can have one gender.
- Member and Orientation will have a one-to-many relationship, where many members can have one orientation.
- Member and RelationshipInterest will have a one-to-many relationship, where many members can have one relationship interest.
- Member and Message will have a many-to-many relationship, where the many-to-many relationship is resolved through having two member ID's - for the sender and for the receiver - thus making the relationship into separate one-to-many relationships, where one member can receive many messages, and one member can send many messages.

Identify the attribute domains for the database:

- Gender
  - Male
  - Female
  - Non-Binary
  - Transgender - Male to Female
  - Transgender - Female to Male
- Orientation
  - Straight
  - Bisexual
  - Gay
  - Lesbian
  - Pansexual
- RelationshipInterest
  - Casual
  - Monogamy
  - Polyamory
  - Non-monogamy
  - Long term relationship

All relevant attributes to be used in the database are identified and discussed further in Table 3 through to Table 9.

Tables in the database with their associated primary and foreign keys:

Entity	Primary Key	Foreign Key
Member	memberID	gender, orientation, relationshipInterest
Login	memberID	memberID
Match	memberID_1, memberID_2	memberID_1, memberID_2
RelationshipInterest	relationshipInterest_ID	relationshipInterest_ID
Message	messageID	receiverID, senderID
Gender	genderID	genderID
Orientation	orientationID	orientationID

Table 2 - Primary and foreign keys for the TechDating System.

The following stored procedures are used within the database:

- create_match - a stored procedure to match two users and insert them into the matches table on the TechDating database.
- get_profile - a stored procedure to display the users first name, location, age, gender, orientation and interested in data on their profile - this joins data from the member table, the gender table, the orientation table, and the relationship interest table.

The following triggers are used within the database:

- check_age - a trigger to check that a user is over the age of 18 when inserting a new record into the member table.

The following indexes are used within the database:

- Gender
- Orientation
- Relationship Interest
- Username

The above indexes are discussed further in Question 1.1.C. INDEXES.

QUESTION 1.1.A.

A conceptual model of a database includes the important entities that describe the data of a system and the relationships between these entities, in a conceptual model no attributes or primary/foreign keys are specified (1 Key Data, 2018).

The following diagram shows the conceptual model for the TechDating System, as per the question the conceptual design will include the relevant integrity constraints through key declarations and cardinality mapping using Crow's Foot notation (Zaiane, 1998) (Rayan, 2015):

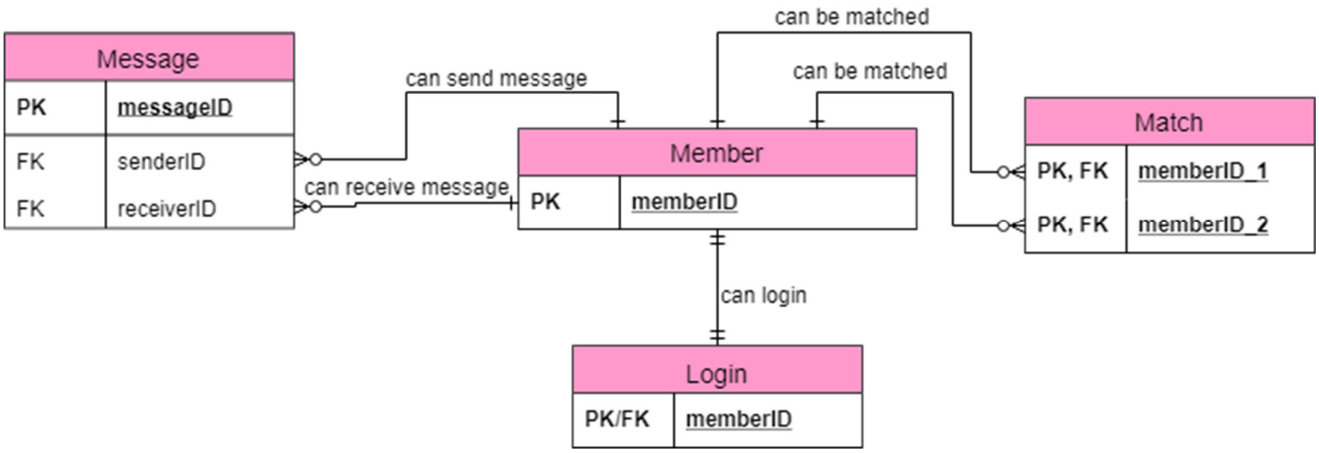

Figure 1 - 0NF Conceptual Model for the TechDating System.

The following diagram further expands on the above conceptual diagram, with the addition of normalisation to the third form:

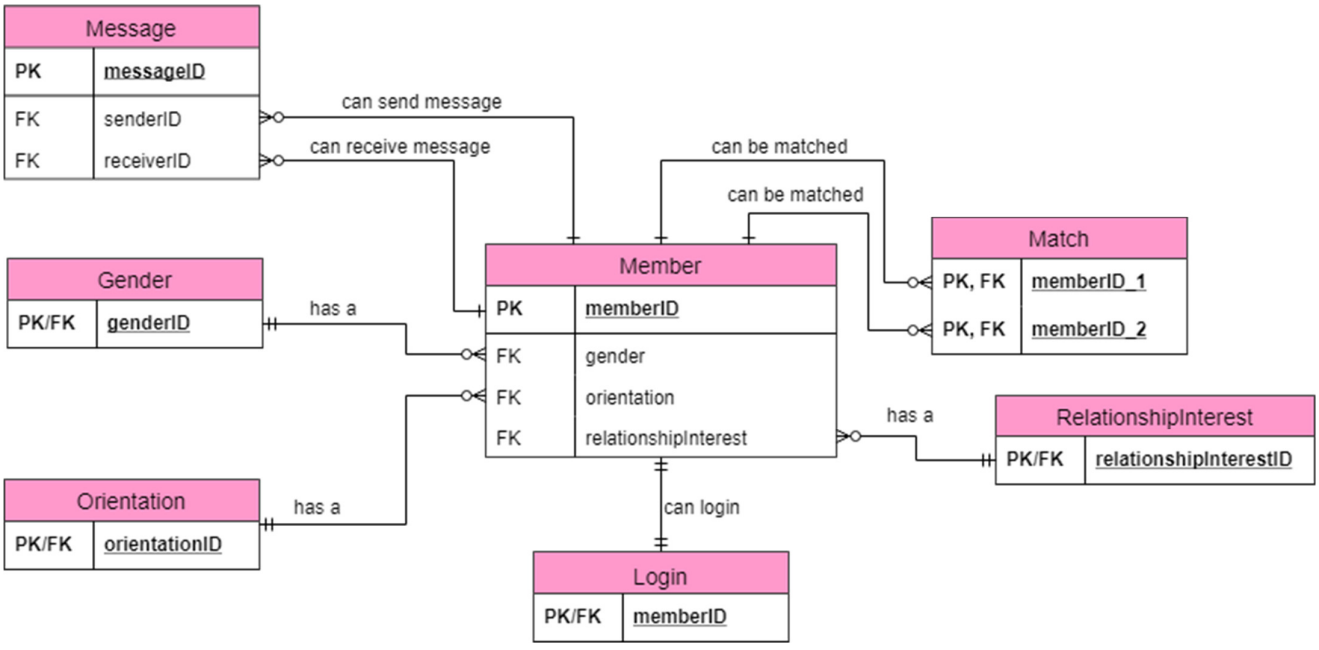

Figure 2 - 3NF Conceptual Model for TechDating System Database.

Figure 2 - 3NF Conceptual Model for TechDating System Database.

QUESTION 1.1.B.

A logical model of a database outlines the database detail without showing how the database will be physically implemented, this includes showing the all entities within the database, the relationships between entities, all attributes in each entity, the foreign and primary keys of each entity, as well as showing normalisation (1 Key Data, 2018).

The following diagram shows the logical model of the TechDating System, showing all of the above-mentioned details such as attributes, primary and foreign keys, relationships between entities using Crow's Foot notation, and the tables have been normalised to third form (Rayan, 2015):

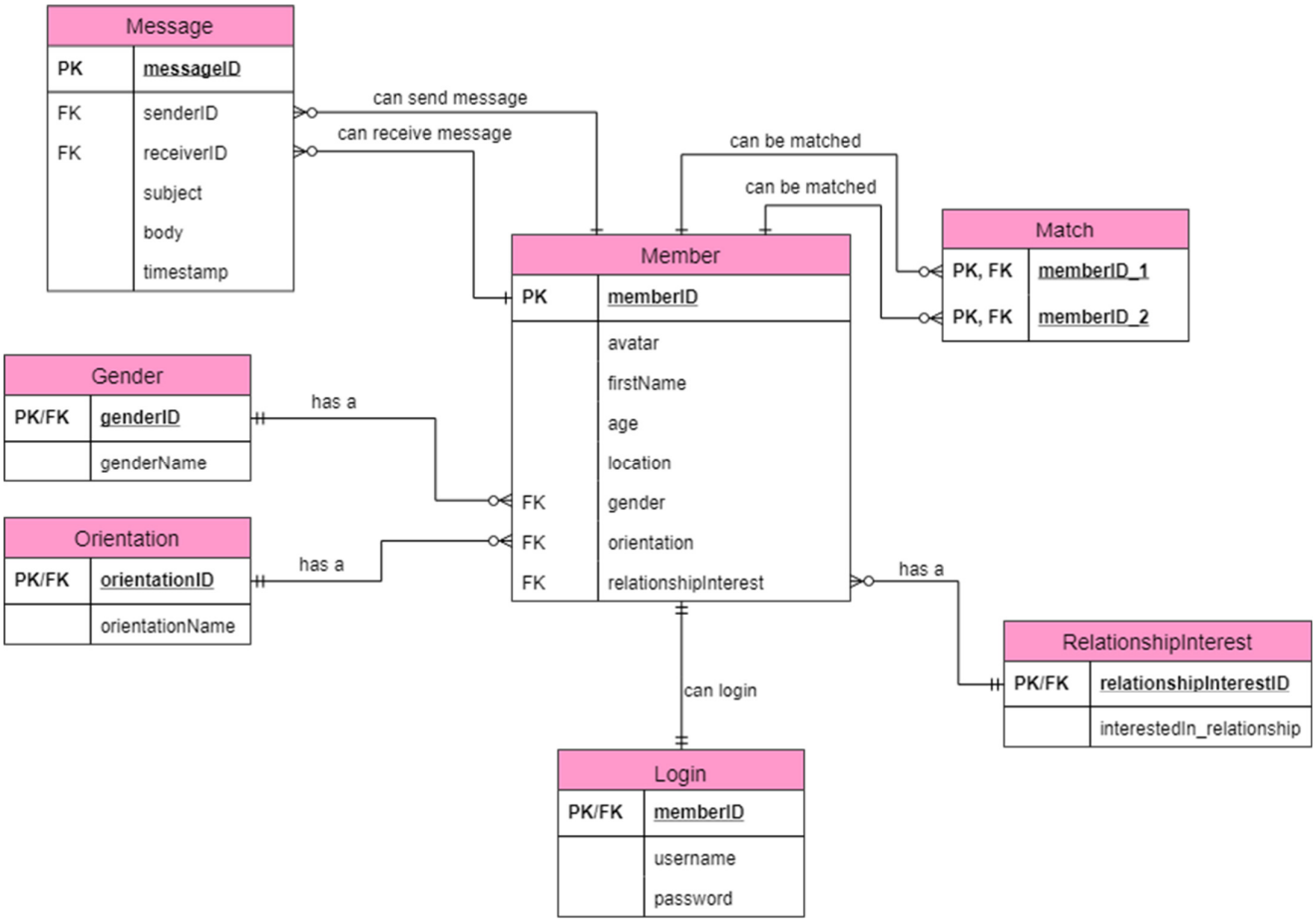

Figure 3 - Logical Model for the TechDating System.

Figure 3 - Logical Model for the TechDating System.

The physical data model of a system shows how the model will be implemented in a database, thus showing all table structures - such as the column name, data type, constraints, primary keys, foreign keys, composite keys, and relationships between the tables (1 Key Data, 2018). When designing the physical model, the following must take place: entities should be converted into tables, relationships should be converted to foreign keys, attributes should be converted to columns, and the physical data model may differ from the logical data model depending on constraints or requirements for the system (1 Key Data, 2018).

The following tables and physical model has been designed for use with MySQL Workbench, thus uses the datatypes for MySQL:

The tables below are used for planning purposes for the physical model:

Member					
Column Name	Data Type	Key	Default Value	Required	Remarks
memberID	INT	Primary Key	None	Yes	Autoincrement
avatar	LONGBLOB		None	No	
firstName	VARCHAR(30)		None	Yes	
age	INT		None	Yes	
location	VARCHAR(50)		None	Yes	
gender	VARCHAR(10)	Foreign Key	None	Yes	
orientation	VARCHAR(10)	Foreign Key	None	Yes	
relationshipInterest	VARCHAR(10)	Foreign Key	None	Yes	

Table 3 - Details of the Member table to be used in the database.

Login					
Column Name	Data Type	Key	Default Value	Required	Remarks
memberID	INT	Primary Key, Foreign Key	None	Yes	
username	VARCHAR(45)		None	Yes	
password	VARCHAR(45)		None	Yes	

Table 4 - Details of the Login table to be used in the database.

Message					
Column Name	Data Type	Key	Default Value	Required	Remarks
messageID	INT	Primary Key	None	Yes	Autoincrement
senderID	INT	Foreign Key	None	Yes	
receiverID	INT	Foreign Key	None	Yes	
subject	VARCHAR(25)		None	Yes	
body	VARCHAR(400)		None	Yes	
timestamp	DATETIME		None	Yes	

Table 5 - Details of the Message table to be used in the database.

Gender					
Column Name	Data Type	Key	Default Value	Required	Remarks
genderID	VARCHAR(10)	Primary Key, Foreign Key	None	Yes	
genderName	VARCHAR(25)		None	Yes	

Table 6 - Details of the Gender table to be used in the database.

Orientation					
Column Name	Data Type	Key	Default Value	Required	Remarks
orientationID	VARCHAR(10)	Primary Key, Foreign Key	None	Yes	
orientationName	VARCHAR(25)		None	Yes	

Table 7 - Details of the Orientation table to be used in the database.

RelationshipInterest					
Column Name	Data Type	Key	Default Value	Required	Remarks
relationshipInterestID	VARCHAR(10)	Primary Key, Foreign Key	None	Yes	
interestedIn_relationship	VARCHAR(25)		None	Yes	

Table 8 - Details of the RelationshipInterest table to be used in the database.

Match					
Column Name	Data Type	Key	Default Value	Required	Remarks
memberID_1	INT	Primary Key, Foreign Key	None	Yes	
memberID_2	INT	Primary Key, Foreign Key	None	Yes	

Table 9 - Details of the Match table to be used in the database.

The figure below shows the physical model for the TechDating System, showing all tables and columns within the database, as well as primary and foreign keys, relationships and any constraints within the database system (1 Key Data, 2018):

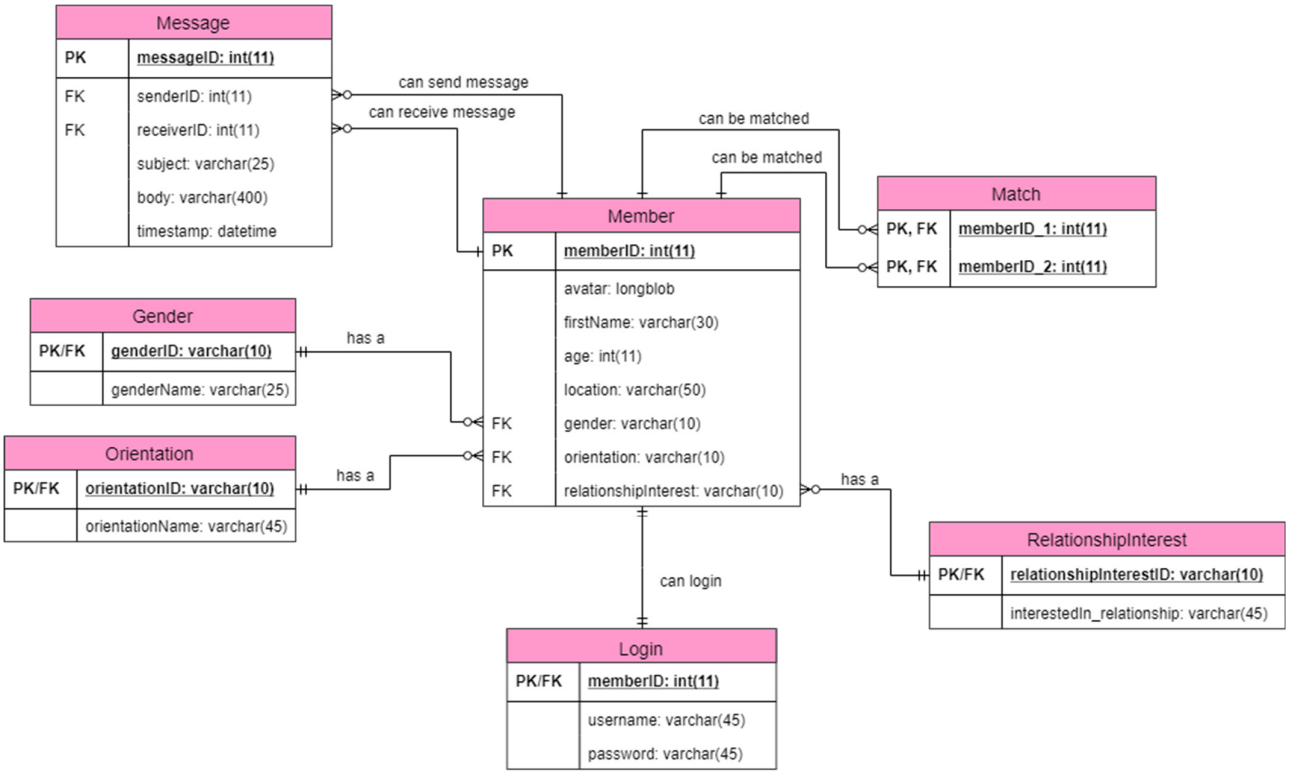

Figure 4 - Physical Model for the TechDating System.

The following diagram is the generated physical model from MySQL Workbench after the database was created:

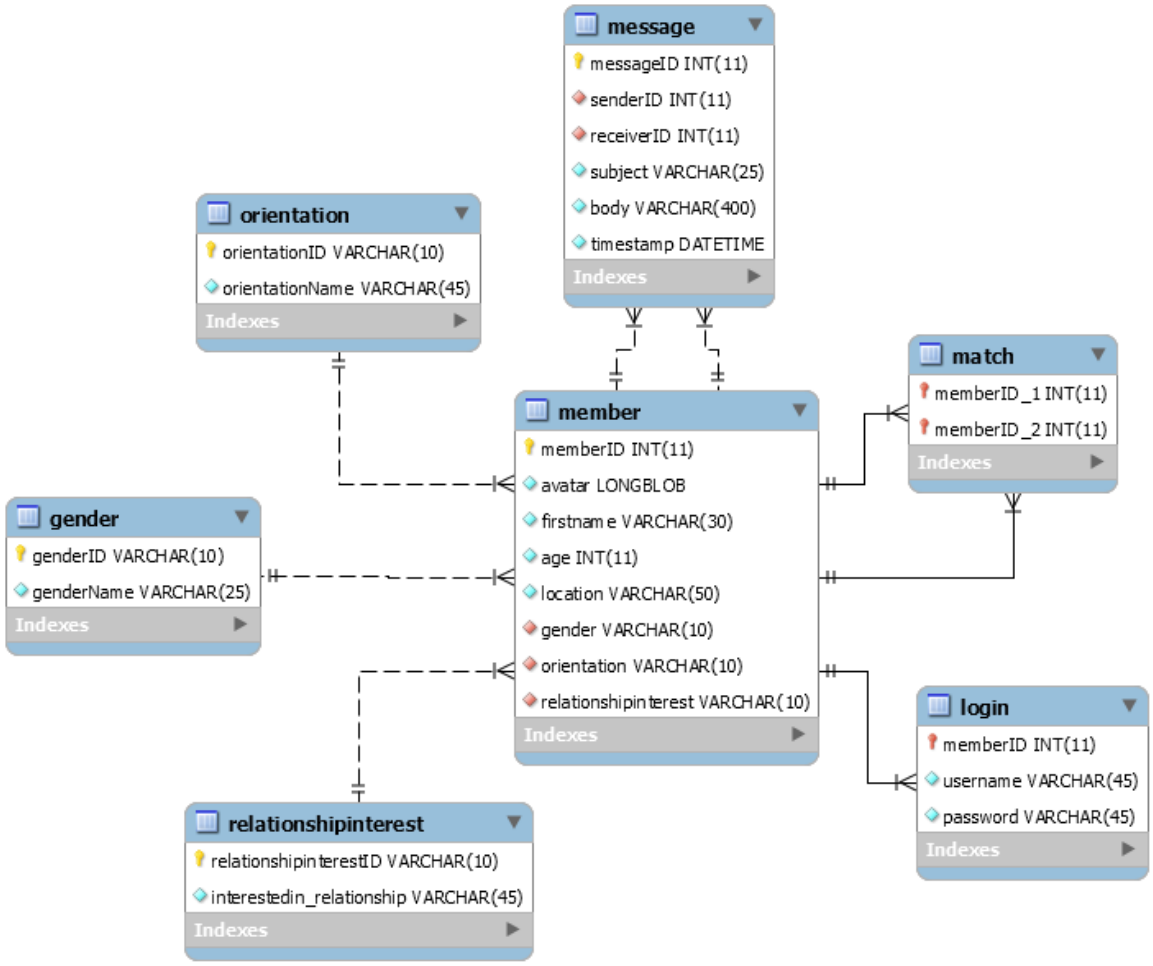

Figure 5 - Physical Model of the TechDating System Database.

Figure 5 - Physical Model of the TechDating System Database. This shows all table names, column names, data types, foreign keys, composite keys, relationships and constraints within the database.

The following diagram shows the physical model with indexes and triggers included:

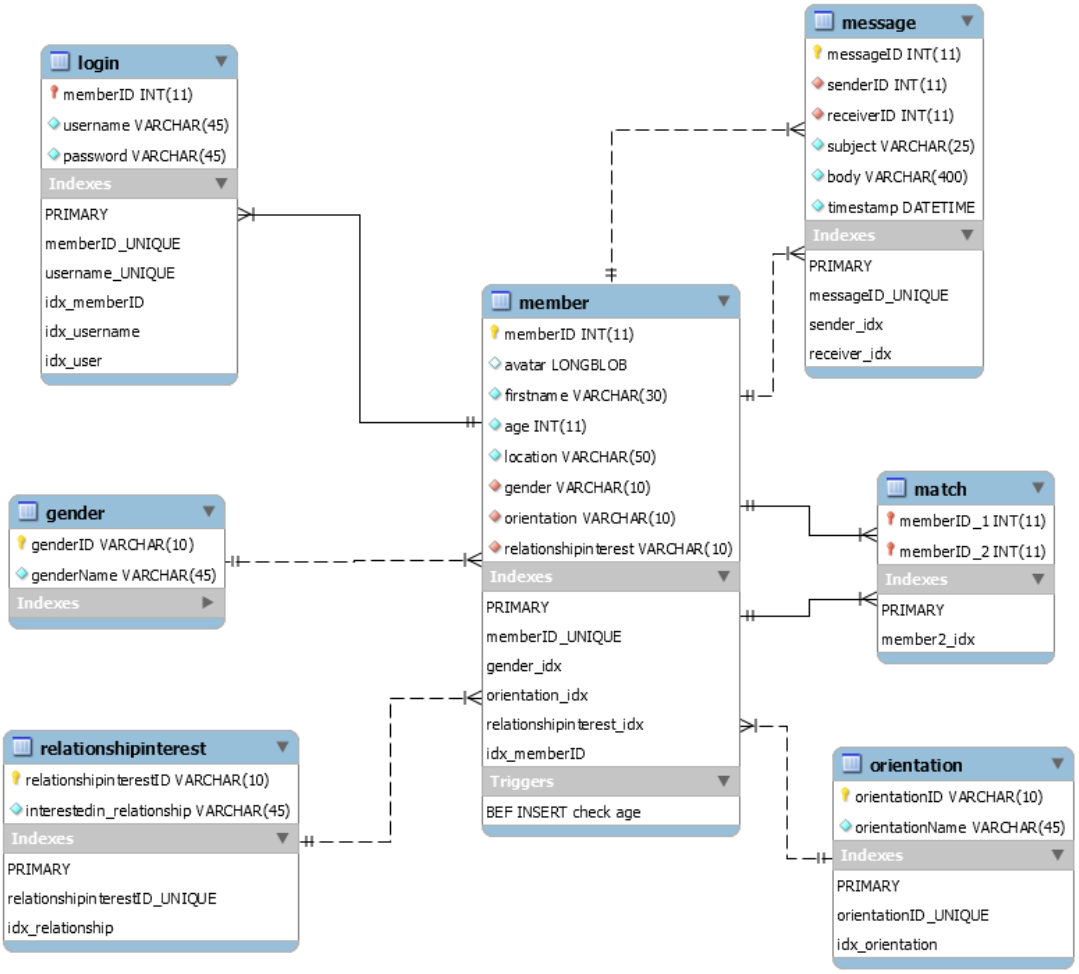

Figure 6 - Physical Model of the TechDating System Database.

Figure 6 shows the final design of the model once triggers and indexes have been added to the DBMS.

The physical model of the database is always the most complex, providing enough information for the developer to create the database and implement it in the system (1 Key Data, 2018). The physical model represents the platform specific - in this case MySQL Workbench - schema of the database, thus the datatypes are designed to be implemented on a MySQL Server and will not work if implemented into any other DBMS platform such as Oracle, without the sufficient modifications made to it (Rayan, 2015). The physical model is an actual representation of the database model and using the correct tools on a DBMS software a database can be automatically generated from an imported physical model design (Rayan, 2015). It is important to note that any many-to-many relationships should be resolved before implementing the physical model as it is not supported in most DBMS software (Rayan, 2015).

QUESTION 1.1.C.

The following section outlines the creation of a functional database.

CREATION OF THE DATABASE

The following SQL code was used to create the TechDating Database:

```
1   CREATE SCHEMA `techdatingdb` ;
```

Figure 7 - SQL script to create the database.

CREATION OF TABLES

MEMBER TABLE

```
CREATE TABLE `member` (
  `memberID` int(11) NOT NULL AUTO_INCREMENT,
  `avatar` longblob,
  `firstname` varchar(30) NOT NULL,
  `age` int(11) NOT NULL,
  `location` varchar(50) NOT NULL,
  `gender` varchar(10) NOT NULL,
  `orientation` varchar(10) NOT NULL,
  `relationshipinterest` varchar(10) NOT NULL,
  PRIMARY KEY (`memberID`),
  UNIQUE KEY `memberID_UNIQUE` (`memberID`),
  KEY `gender_idx` (`gender`),
  KEY `orientation_idx` (`orientation`),
  KEY `relationshipinterest_idx` (`relationshipinterest`),
  CONSTRAINT `gender` FOREIGN KEY (`gender`) REFERENCES `gender` (`genderID`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `orientation` FOREIGN KEY (`orientation`) REFERENCES `orientation` (`orientationID`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `relationshipinterest` FOREIGN KEY (`relationshipinterest`) REFERENCES `relationshipinterest` (`relationshipinterestID`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8 COMMENT='Database to store all members of TechDating.';

```

Figure 8 - SQL script to create the member table.

```
INSERT INTO `techdatingdb`.`member`
(`avatar`,
`firstname`,
`age`,
`location`,
`gender`,
`orientation`,
`relationshipinterest`)
VALUES
(load_file('C:\Users\Sydney Twigg\Documents\CTI\Third Year\ITDA310\Project\Images\Sydney.jpg'),
'Sydney',
'21',
'Cape Town, South Africa',
'F',
'S',
'LTR');

```

Figure 9 - SQL script inserting test data into the member table.

memberID	avatar	firstname	age	location	gender	orientation	relationshipinterest
1	NULL	Sydney	21	Cape Town, South Africa	F	S	LTR

Figure 10 - Screenshot showing the successful entry of data into the member table.

LOGIN TABLE

```
CREATE TABLE `login` (
  `memberID` int(11) NOT NULL,
  `username` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  PRIMARY KEY (`memberID`),
  UNIQUE KEY `memberID_UNIQUE` (`memberID`),
  UNIQUE KEY `username_UNIQUE` (`username`),
  CONSTRAINT `memberID` FOREIGN KEY (`memberID`) REFERENCES `member` (`memberID`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Table to store all login details for TechDating';

```

Figure 11 - SQL script to create the login table

```
INSERT INTO `techdatingdb`.`login`
(`memberID`,
`username`,
`password`)
VALUES
('1',
'Syd',
'Password1');

```

Figure 12 - SQL script to insert test data into the login table.

	memberID	username	password
	1	Syd	Password1

Figure 13 - Screenshot showing the successful entry of data into the login table.

GENDER TABLE

```sql
CREATE TABLE `gender` (
  `genderID` varchar(10) NOT NULL,
  `genderName` varchar(45) NOT NULL,
  PRIMARY KEY (`genderID`),
  UNIQUE KEY `genderID_UNIQUE` (`genderID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Table to store the list of accepted gender inputs';
```

Figure 14 - SQL script to create the gender table.

```sql
INSERT INTO `techdatingdb`.`gender`
(`genderID`,
`genderName`)
VALUES
('M',
'Male');

INSERT INTO `techdatingdb`.`gender`
(`genderID`,
`genderName`)
VALUES
('F',
'Female');

INSERT INTO `techdatingdb`.`gender`
(`genderID`,
`genderName`)
VALUES
('NB',
'Non-Binary');

INSERT INTO `techdatingdb`.`gender`
(`genderID`,
`genderName`)
VALUES
('MTF',
'Transgender - Male to Female');

INSERT INTO `techdatingdb`.`gender`
(`genderID`,
`genderName`)
VALUES
('FTM',
'Transgender - Female to Male');
```

Figure 15 - SQL script to enter the attribute domain values for the gender table.

genderID	genderName
F	Female
FTM	Transgender - Female to Male
M	Male
MTF	Transgender - Male to Female
NB	Non-Binary

Figure 16 - Screenshot showing the successful entry of data into the gender table.

ORIENTATION TABLE

```sql
CREATE TABLE `orientation` (
  `orientationID` varchar(10) NOT NULL,
  `orientationName` varchar(45) NOT NULL,
  PRIMARY KEY (`orientationID`),
  UNIQUE KEY `orientationID_UNIQUE` (`orientationID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Table to store the list of accepted orientation inputs';
```

Figure 17 - SQL script to create the orientation table.

```sql
INSERT INTO `techdatingdb`.`orientation`
(`orientationID`,
`orientationName`)
VALUES
('S',
'Straight');

INSERT INTO `techdatingdb`.`orientation`
(`orientationID`,
`orientationName`)
VALUES
('G',
'Gay');

INSERT INTO `techdatingdb`.`orientation`
(`orientationID`,
`orientationName`)
VALUES
('L',
'Lesbian');

INSERT INTO `techdatingdb`.`orientation`
(`orientationID`,
`orientationName`)
VALUES
('B',
'Bisexual');

INSERT INTO `techdatingdb`.`orientation`
(`orientationID`,
`orientationName`)
VALUES
('P',
'Pansexual');
```

Figure 18- SQL script to enter the attribute domain values for the orientation table.

orientationID	orientationName
B	Bisexual
G	Gay
L	Lesbian
P	Pansexual
S	Straight

Figure 19 - Screenshot showing the successful entry of data into the orientation table.

RELATIONSHIP INTEREST TABLE

```sql
CREATE TABLE `relationshipinterest` (
  `relationshipinterestID` varchar(10) NOT NULL,
  `interestedin_relationship` varchar(45) NOT NULL,
  PRIMARY KEY (`relationshipinterestID`),
  UNIQUE KEY `relationshipinterestID_UNIQUE` (`relationshipinterestID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Table to store the list of accepted relationship interest inputs';
```

Figure 20 - SQL script to create the relationship interest table.

```sql
INSERT INTO `techdatingdb`.`relationshipinterest`
(`relationshipinterestID`,
`interestedin_relationship`)
VALUES
('C',
'Casual');

INSERT INTO `techdatingdb`.`relationshipinterest`
(`relationshipinterestID`,
`interestedin_relationship`)
VALUES
('M',
'Monogamy');

INSERT INTO `techdatingdb`.`relationshipinterest`
(`relationshipinterestID`,
`interestedin_relationship`)
VALUES
('P',
'Polyamory');

INSERT INTO `techdatingdb`.`relationshipinterest`
(`relationshipinterestID`,
`interestedin_relationship`)
VALUES
('NM',
'Non-Monogamy');

INSERT INTO `techdatingdb`.`relationshipinterest`
(`relationshipinterestID`,
`interestedin_relationship`)
VALUES
('LTR',
'Long Term Relationship');
```

Figure 21 - SQL script to enter the attribute domain values for the relationship interest table.

relationshipinterestID	interestedin_relationship
C	Casual
LTR	Long Term Relationship
M	Monogamy
NM	Non-Monogamy
P	Polyamory

Figure 22 - Screenshot showing the successful entry of data into the relationship interest table.

MATCH TABLE

```
CREATE TABLE `match` (
  `memberID_1` int(11) NOT NULL,
  `memberID_2` int(11) NOT NULL,
  PRIMARY KEY (`memberID_1`,`memberID_2`),
  KEY `member2_idx` (`memberID_2`),
  CONSTRAINT `member1` FOREIGN KEY (`memberID_1`) REFERENCES `member` (`memberID`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `member2` FOREIGN KEY (`memberID_2`) REFERENCES `member` (`memberID`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Table to store the ID''s of two matches users';
```

Figure 23 - SQL script to create the match table.

MESSAGES TABLE

```
CREATE TABLE `message` (
  `messageID` int(11) NOT NULL AUTO_INCREMENT,
  `senderID` int(11) NOT NULL,
  `receiverID` int(11) NOT NULL,
  `subject` varchar(25) NOT NULL,
  `body` varchar(400) NOT NULL,
  `timestamp` datetime NOT NULL,
  PRIMARY KEY (`messageID`),
  UNIQUE KEY `messageID_UNIQUE` (`messageID`),
  KEY `sender_idx` (`senderID`),
  KEY `receiver_idx` (`receiverID`),
  CONSTRAINT `receiver` FOREIGN KEY (`receiverID`) REFERENCES `member` (`memberID`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `sender` FOREIGN KEY (`senderID`) REFERENCES `member` (`memberID`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Table to store messages sent between user''s';
```

Figure 24 - SQL to create the message table.

STORED PROCEDURES

A stored procedure is a subroutine that is stored within a database; a procedure has a name, parameter list and SQL statements to carry out its functionality (W3Resource, 2018). Stored procedures must be invoked using a call statement. Stored procedures are used because they are fast, portable, migratory and always available as source code in the database itself (W3Resource, 2018).

The following stored procedures were created for the TechDating database:

CREATE MATCH

```
USE `techdatingdb`;
DROP PROCEDURE IF EXISTS `create_match`;

DELIMITER //
CREATE PROCEDURE `create_match`
(
        /*The input parameters for this procedure are the matched
        member IDs.*/
        IN `memberID_1` INTEGER,
        IN `memberID_2` INTEGER
)
BEGIN
    /*Two users will be automatically matched together using
    an algorithm in the Java code. This procedure is then called
    using their ID's as parameters to place those users into the
    match table.*/
    INSERT INTO `techdatingdb`.`match`(`memberID_1`, `memberID_2`)
    VALUES (`memberID_1`, `memberID_2`);
END //


DELIMITER;
```

Figure 25 - SQL script to create the procedure create_match.

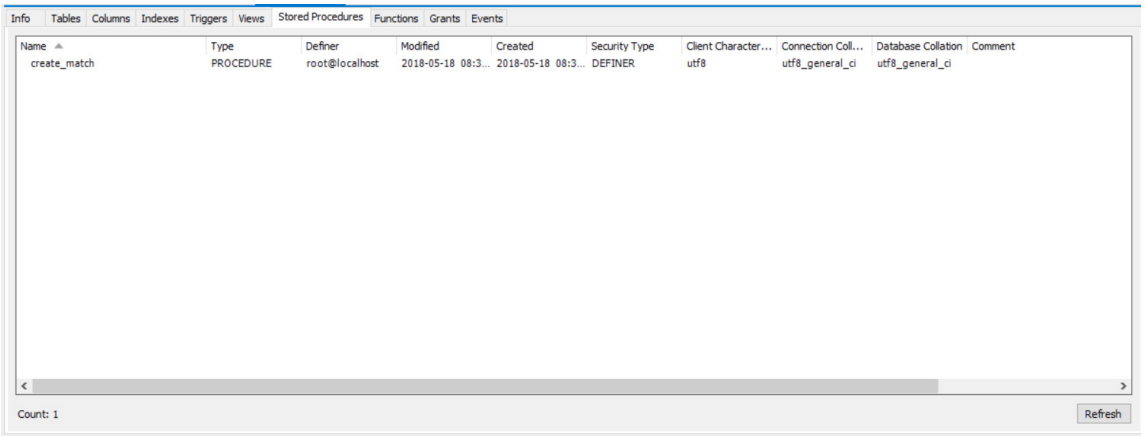

Figure 26 - Screenshot showing the successful creation of the create_match stored procedure.

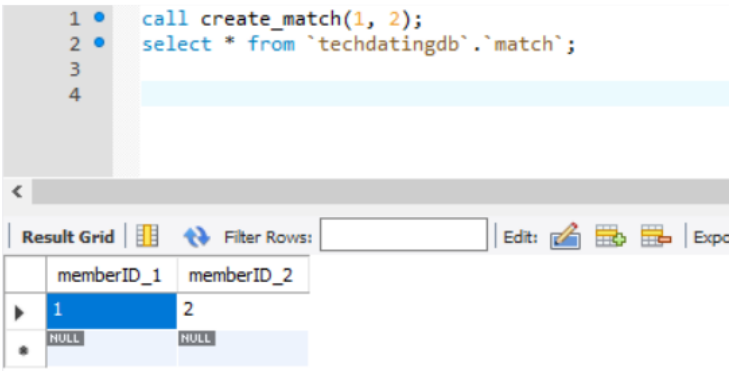

Figure 27 - SQL script and screenshot showing the successful execution of the create_match procedure.

```
USE `techdatingdb`;
DROP PROCEDURE IF EXISTS `get_profile`;

DELIMITER //
CREATE PROCEDURE `get_profile`
(
        /*The input parameter for this procedure is the
        logged in member IDs.*/
        IN `memberID` INTEGER
)
BEGIN
    /*When the member logs in, their profile details
    will be fetched from the database in the member
    table - this makes use of inner joins to fetch the name
    of the gender, orientation and relationship interest from
    their ID's.*/
    SELECT member.firstname, member.age, member.location, gender.genderName,
    orientation.orientationName, relationshipinterest.interestedin_relationship
    FROM (((member
    INNER JOIN gender ON member.gender=gender.genderID)
    INNER JOIN orientation on member.orientation=orientation.orientationID)
    INNER JOIN relationshipInterest on
    member.relationshipinterest=relationshipinterest.relationshipinterestID);
END //


DELIMITER;
```

Figure 28 - SQL script to create the get_profile stored procedure.

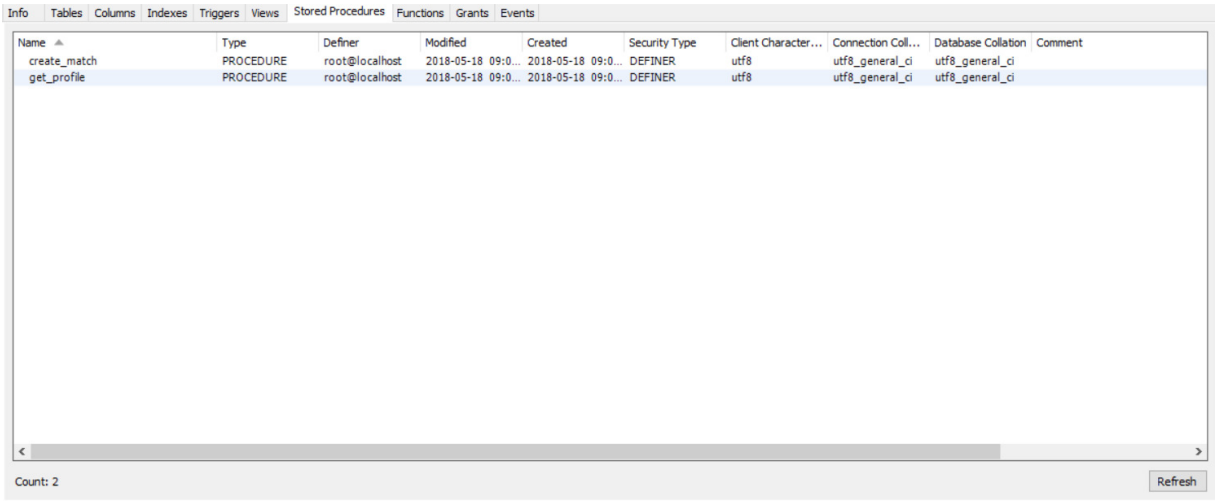

Figure 29 - Screenshot showing the successful creation of the get_profile stored procedure

Figure 30 - SQL script and screenshot showing the successful execution of the get_profile procedure.

TRIGGERS

Triggers are a type of stored procedure within a database, that runs when specific actions occur; triggers can be created to run after or before database actions such as insert, update, and delete (Wenzel, 2018).

Triggers will be used to validate a user's age in the TechDating System, as users must be over the age of 18 to register on the system. It is better to use database level validation over application level validation, as a database can be used by multiple applications or interfaces - where controlling data consistency and validity becomes harder to manage, therefore when the validation is programmed at database level the data integrity and consistency will be maintained (Vuorinen, 2013).

CHECK AGE

```
1  USE `techdatingdb`;
2  DROP TRIGGER IF EXISTS `check_age`;
3
4  DELIMITER //
5  CREATE TRIGGER `check age`
6  BEFORE INSERT ON `techdatingdb`.`member` FOR EACH ROW
7  /*Trigger to check the age of a user, users must be
8  over the age of 18 to register*/
9  BEGIN
10     IF(NEW.`age` < '18') THEN
11         SIGNAL SQLSTATE '45000'
12         SET MESSAGE_TEXT = 'Users must be over 18.';
13     END IF;
14 END//
15
16
```

Figure 31 - SQL script to create the check_age trigger.

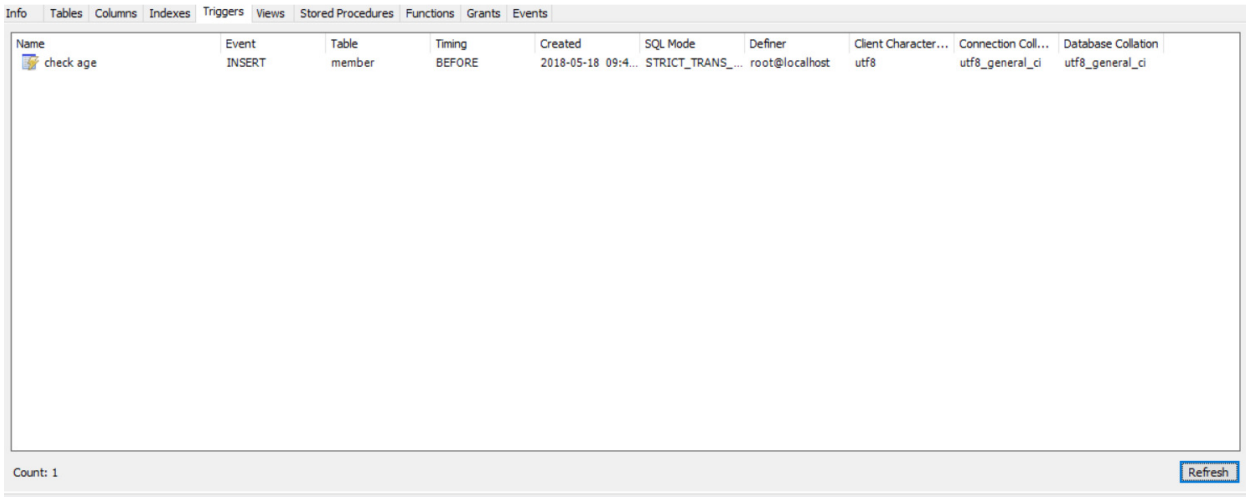

Figure 32 - Screenshot showing the trigger was successfully created.

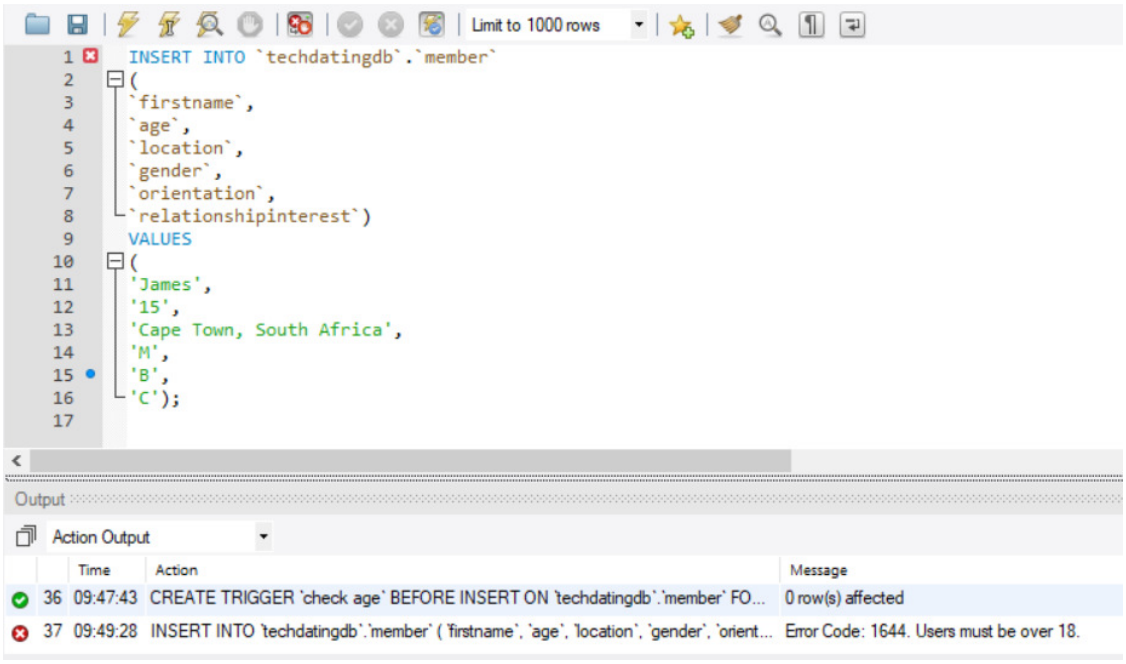

Figure 33 - SQL script and screenshot showing the check_age trigger is functioning correctly.

INDEXES

Indexes are used to speed up searches and queries within a database, they assist in retrieving data from the data at a speed (W3 Schools, 2018).

GENDER

An index will be created for the gender table as the ID's will be used to search for the gender names from the member table.

```
2  CREATE INDEX idx_gender
3  ON gender (genderID, genderName);
4
```

Figure 34 - SQL script to create the index idx_gender on the gender table.

ORIENTATION

An index will be created for the orientation table as the ID's will be used to search for the orientation names from the member table.

```
5  CREATE INDEX idx_orientation
6  ON orientation (orientationID, orientationName);
7
```

Figure 35 - SQL script to create the index idx_orientation on the orientation table.

RELATIONSHIP INTEREST

An index will be created for the relationship interest table as the ID's will be used to search for the relationship type names from the member table.

```
8  CREATE INDEX idx_relationship
9  ON relationshipinterest(relationshipinterestID, interestedin_relationship);
10
```

Figure 36 - SQL script to create the index idx_relationship on the relationship interest table.

USERNAME

An index will be made for the username column on the login table as this is used in a lookup query to find the member ID from the username within the system.

```
1  CREATE INDEX idx_user
2  ON login (memberID, username);
3
```

Figure 37 - SQL script to create the index idx_user on the login table.

The screenshot below shows all the indexes on the TechDating database:

Info | Tables | Columns | Indexes | Triggers | Views | Stored Procedures | Functions | Grants | Events

Table	Name	Unique	Index...	Index Comment	Column	Seq in Index	Packed	Collat...	Cardi...	Sub p...	NULL	Comment
member	gender_idx	No	BTREE		gender	1		A	3			
gender	genderID_UNIQUE	Yes	BTREE		genderID	1		A	5			
gender	idx_gender	No	BTREE		genderID	1		A	5			
gender	idx_gender	No	BTREE		genderName	2		A	5			
member	idx_memberID	No	BTREE		memberID	1		A	5			
login	idx_memberID	No	BTREE		memberID	1		A	4			
orientation	idx_orientation	No	BTREE		orientationName	2		A	5			
orientation	idx_orientation	No	BTREE		orientationID	1		A	5			
relationshipinterest	idx_relationship	No	BTREE		interestedin_rela...	2		A	5			
relationshipinterest	idx_relationship	No	BTREE		relationshipinter...	1		A	5			
login	idx_user	No	BTREE		username	2		A	4			
login	idx_user	No	BTREE		memberID	1		A	4			
login	idx_username	No	BTREE		username	1		A	4			
match	member2_idx	No	BTREE		memberID_2	1		A	0			
member	memberID_UNIQUE	Yes	BTREE		memberID	1		A	5			
login	memberID_UNIQUE	Yes	BTREE		memberID	1		A	4			
message	messageID_UNIQUE	Yes	BTREE		messageID	1		A	0			
member	orientation_idx	No	BTREE		orientation	1		A	3			
orientation	orientationID_UNIQUE	Yes	BTREE		orientationID	1		A	5			
match	PRIMARY	Yes	BTREE		memberID_2	2		A	0			
orientation	PRIMARY	Yes	BTREE		orientationID	1		A	5			
relationshipinterest	PRIMARY	Yes	BTREE		relationshipinter...	1		A	5			
match	PRIMARY	Yes	BTREE		memberID_1	1		A	0			
gender	PRIMARY	Yes	BTREE		genderID	1		A	5			
message	PRIMARY	Yes	BTREE		messageID	1		A	0			
login	PRIMARY	Yes	BTREE		memberID	1		A	4			
member	PRIMARY	Yes	BTREE		memberID	1		A	5			
message	receiver_idx	No	BTREE		receiverID	1		A	0			
member	relationshipinterest_idx	No	BTREE		relationshipinter...	1		A	4			
relationshipinterest	relationshipinterestID_UNIQUE	Yes	BTREE		relationshipinter...	1		A	5			
message	sender_idx	No	BTREE		senderID	1		A	0			
login	username_UNIQUE	Yes	BTREE		username	1		A	4			

Count: 32 Refresh

Figure 38 - Screenshot showing all indexes on the TechDating database.

USERS

The system allows for two users - a regular member of the TechDating site and for a system administrator. The users of the system in a database perspective are discussed further on in QUESTION 1.2.A. and QUESTION 1.2.B.

A regular member of the system can perform basic functions on the system and to the database - inserting records when they register for their account, updating records when they change their personal details and viewing records when they check their messages and matches.

An administrator user can perform many more functions on the system and database than the user; in the application environment, an administrator is able to view all users registered on the system, update their records, or delete members.

The member and administrator both login through the login page, the user will be redirected to their profile on successful login while the administrator will be redirected to the administration tools page.

DESIGN PROTOTYPE

The following prototypes were created using MockingBot to design the front-end aspect of the system:

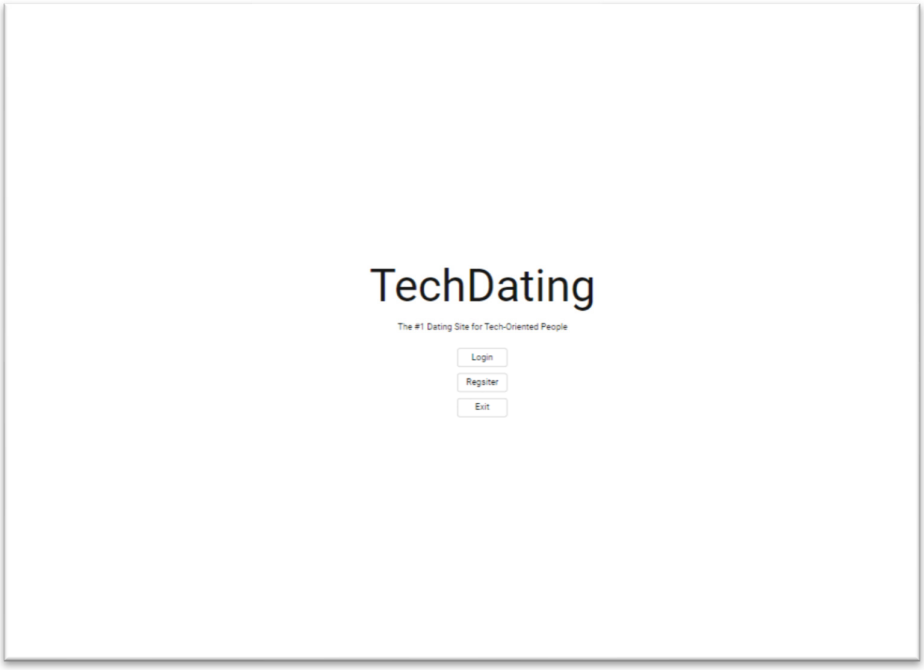

Figure 39 - Prototype user-interface for the Home page.

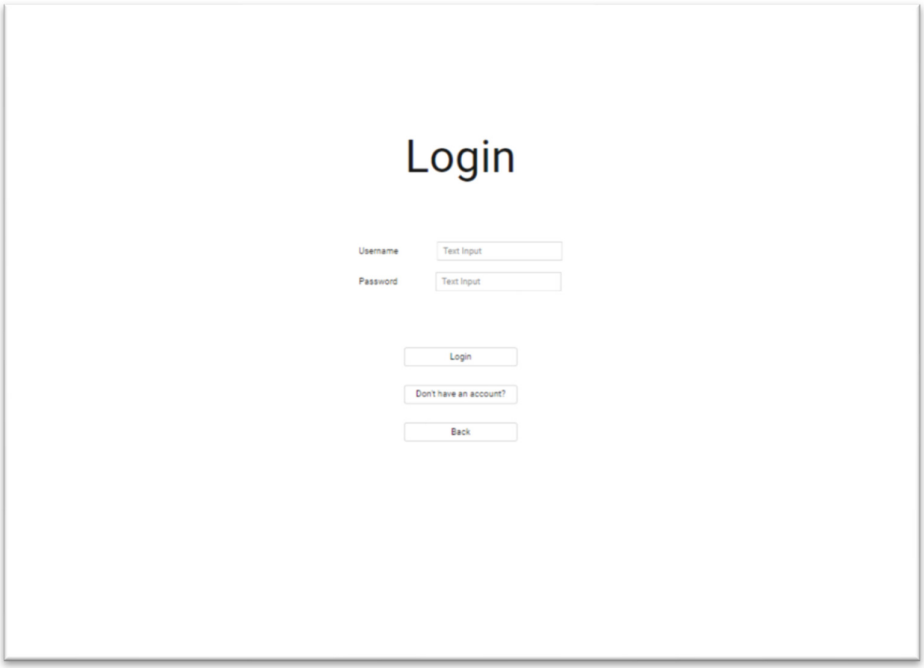

Figure 40 - Prototype user-interface for the Login page.

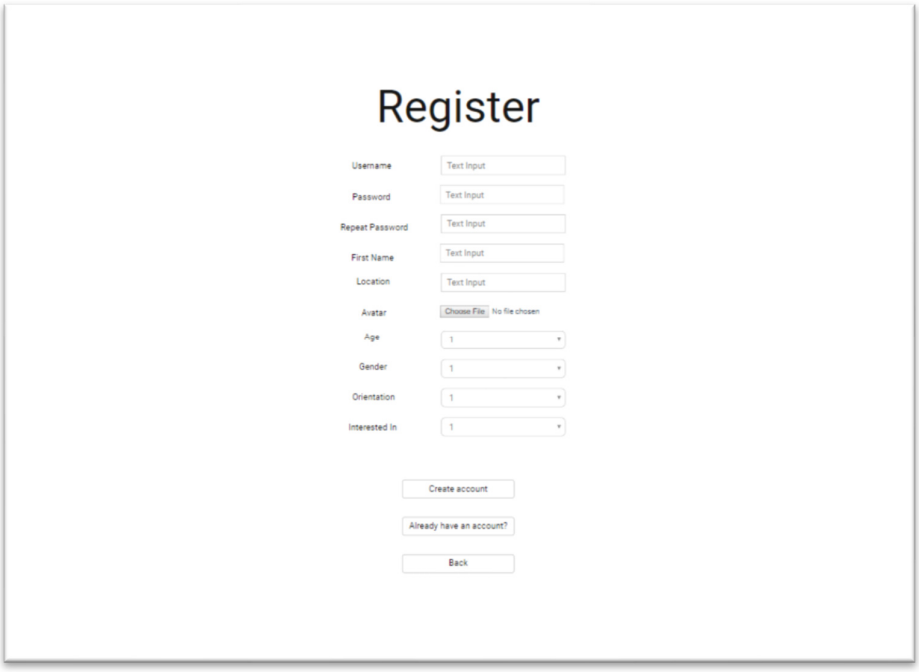

Figure 41 - Prototype user-interface for the Registration page.

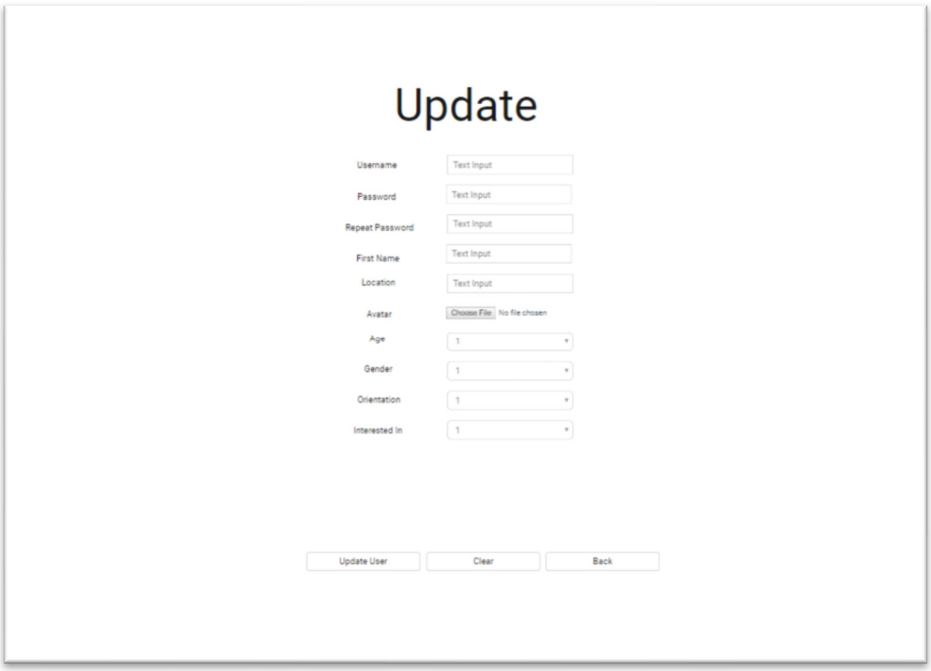

Figure 42 - Prototype user-interface for the Update page.

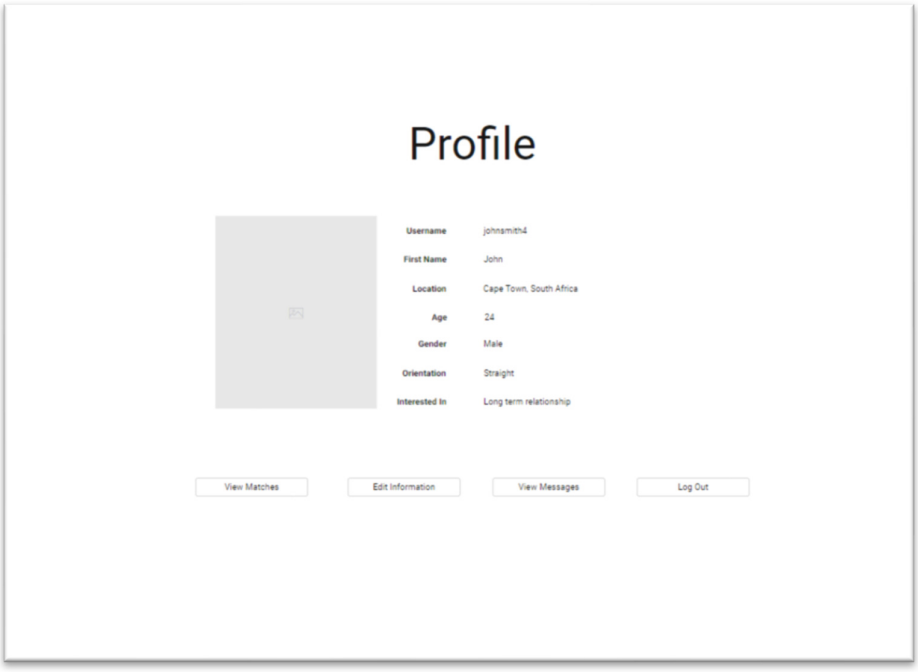

Figure 43 - Prototype user-interface for the Profile page.

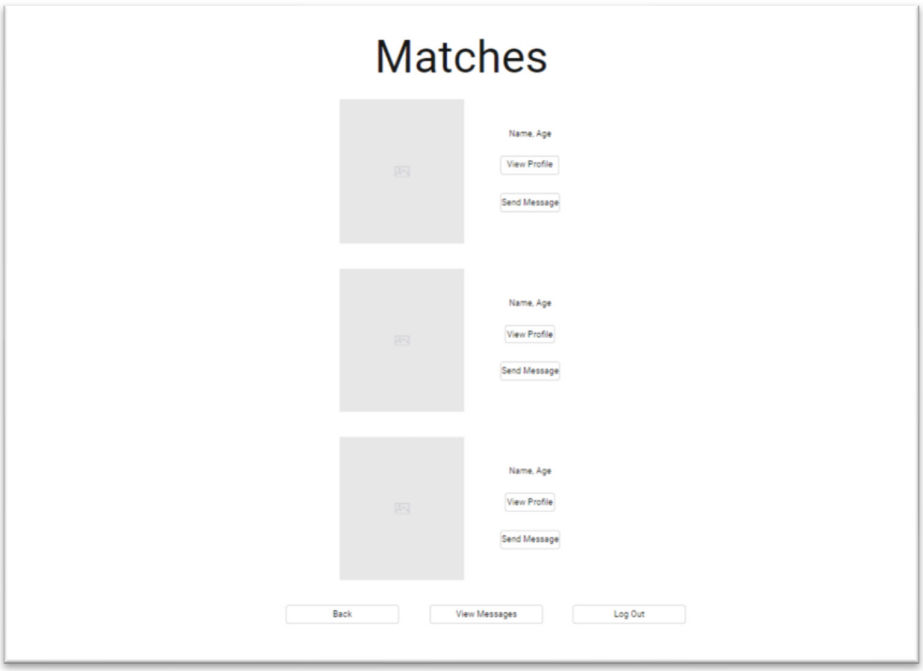

Figure 44 - Prototype user-interface for the Matches page.

Figure 45 - Prototype user-interface for the Messages page.

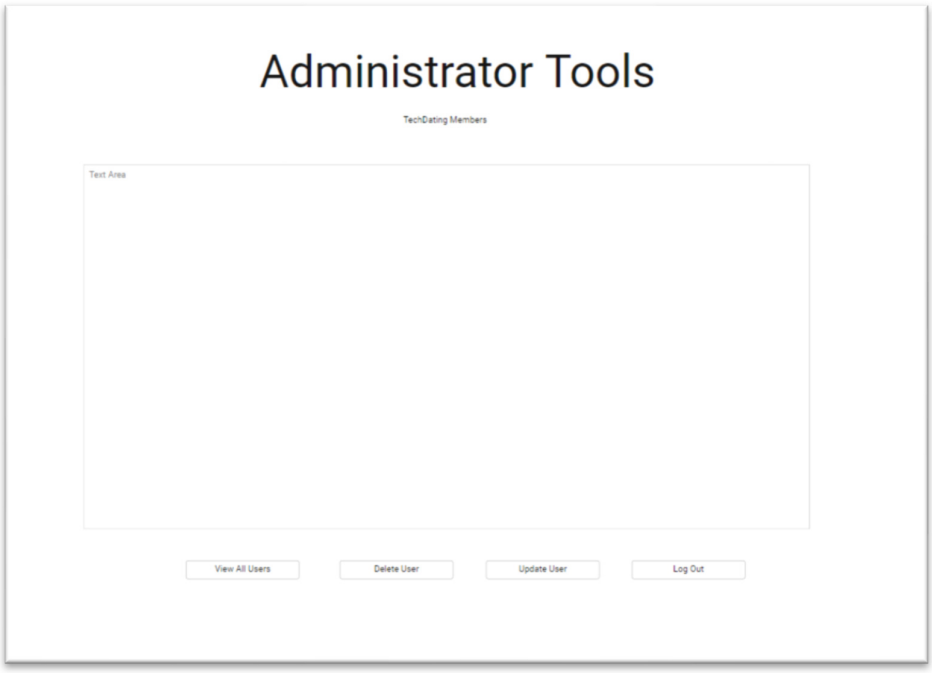

Figure 46 - Prototype user-interface for the Administration page.

FINAL APPLICATION DESIGN

The following screenshots show the final design of the system:

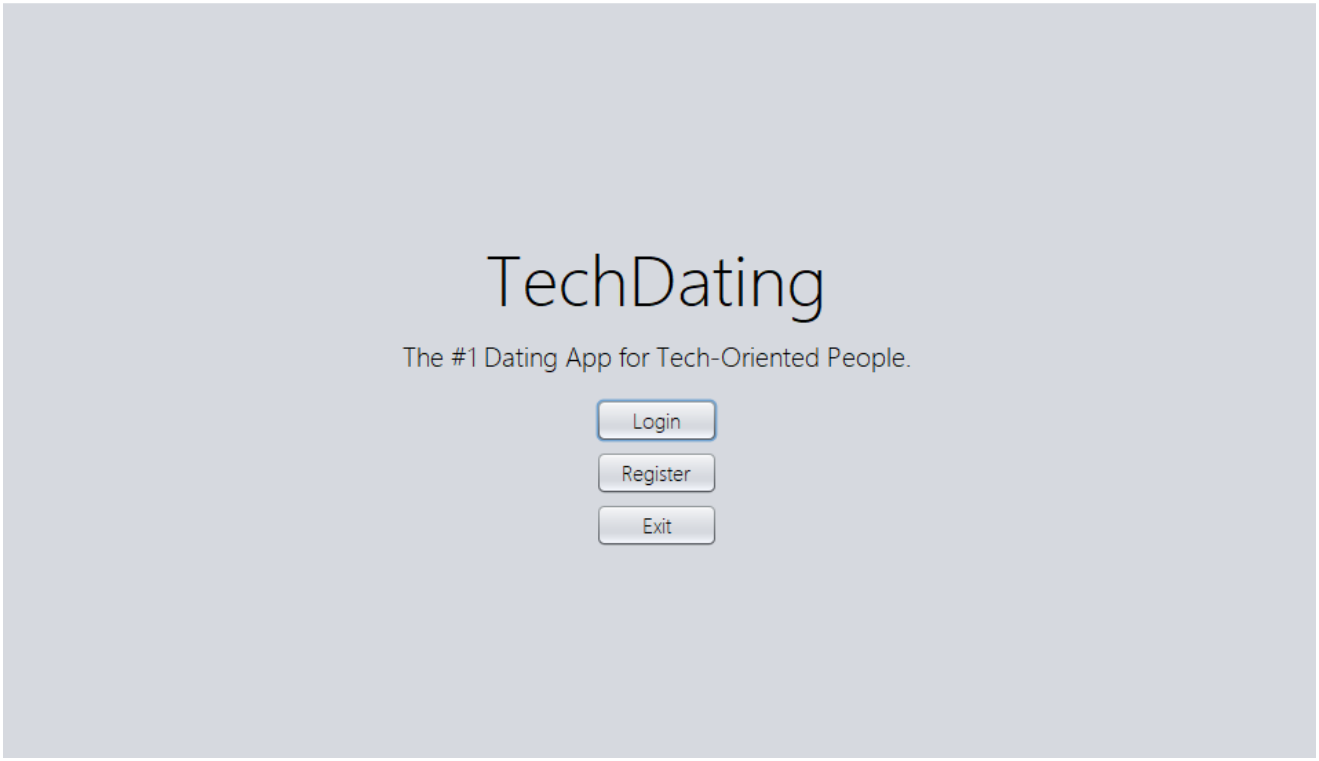

Figure 47 - Screenshot of the TechDating home page.

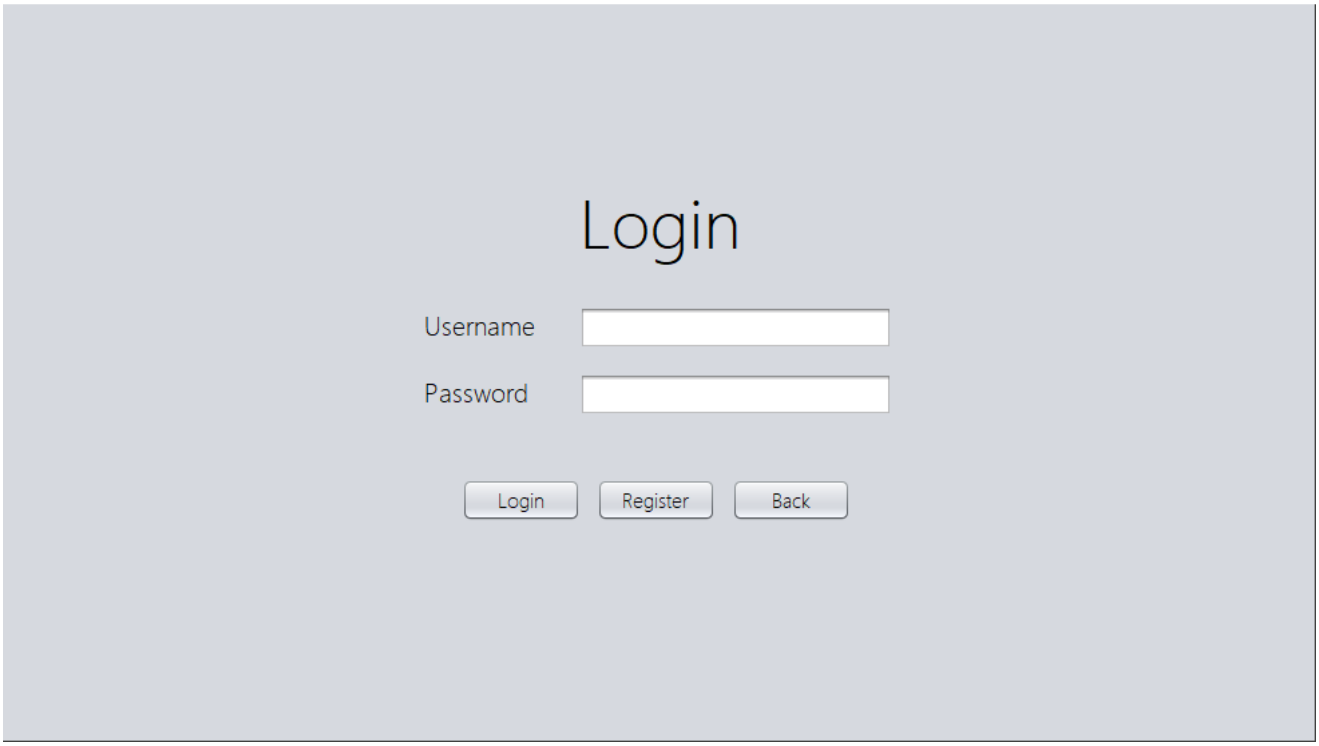

Figure 48 - Screenshot of the TechDating login page.

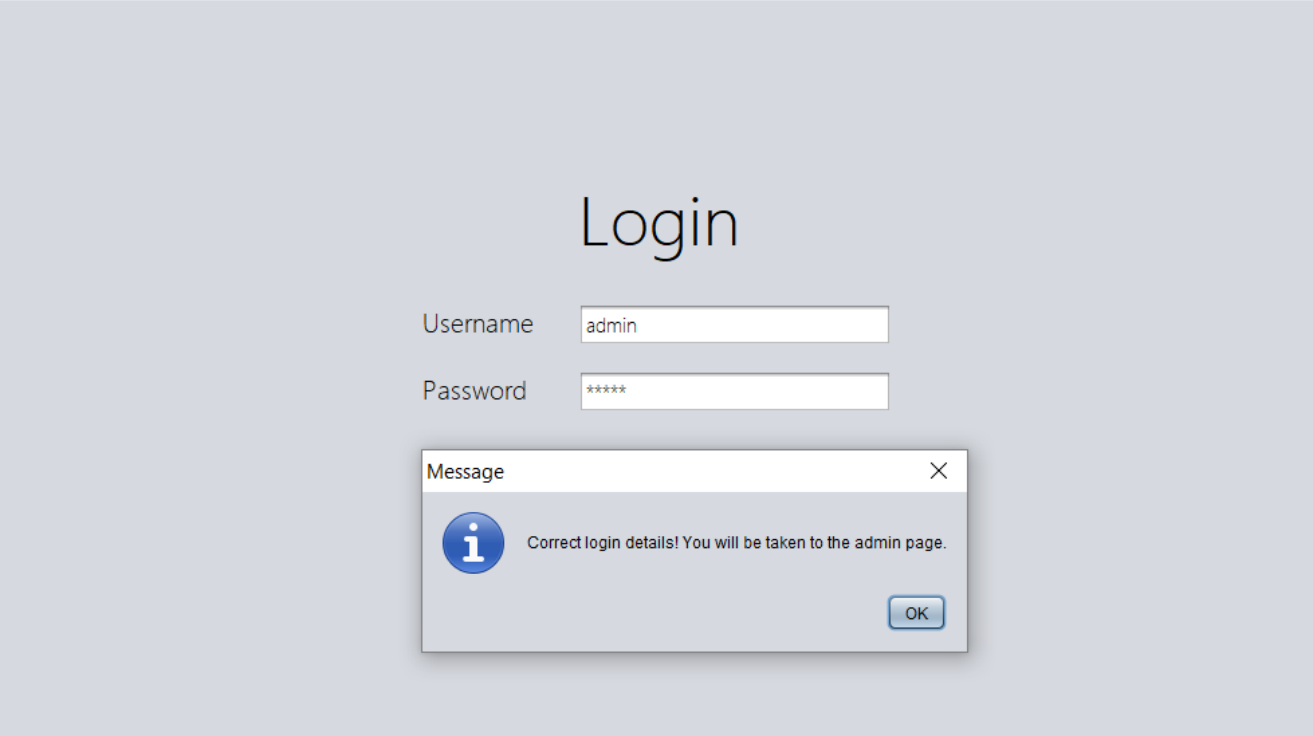

Figure 49 - Screenshot of a successful administrator login message.

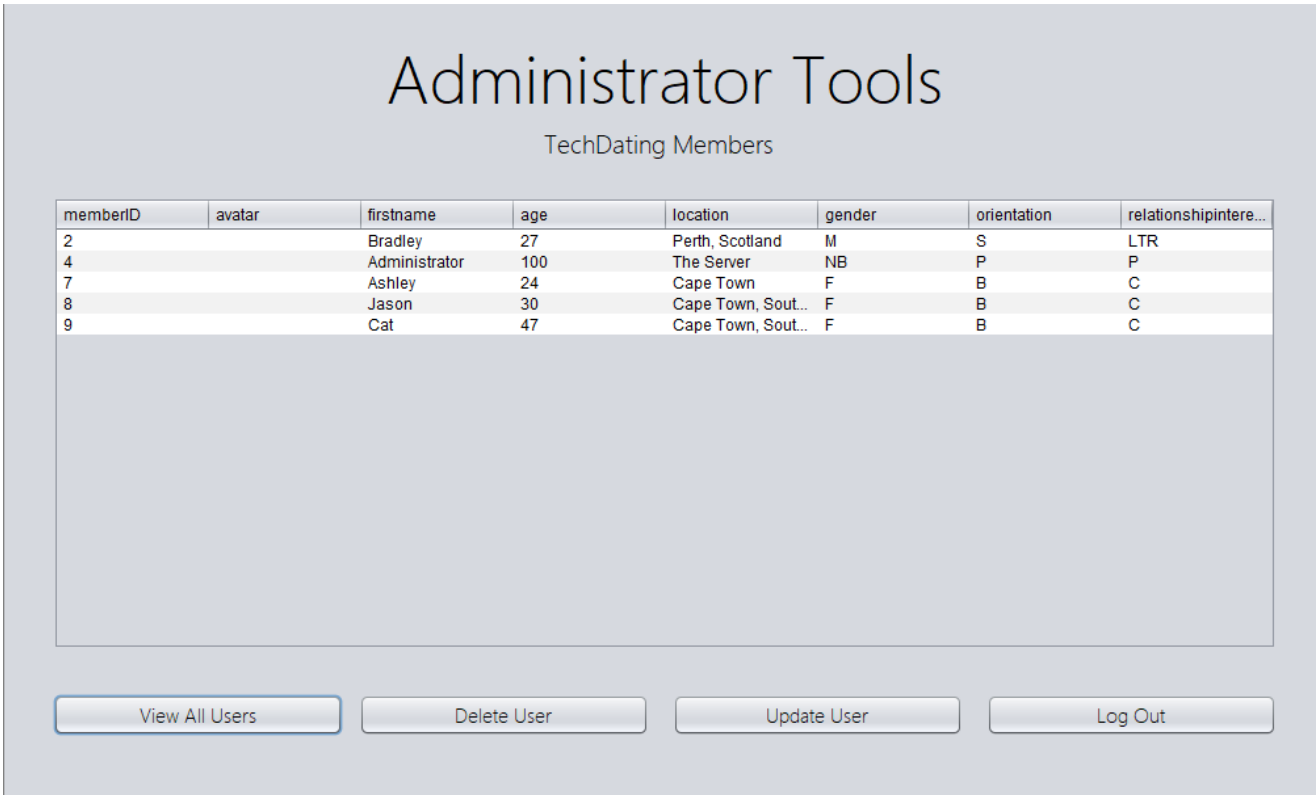

Figure 50 - Screenshot of the administrator tools page of the TechDating system.

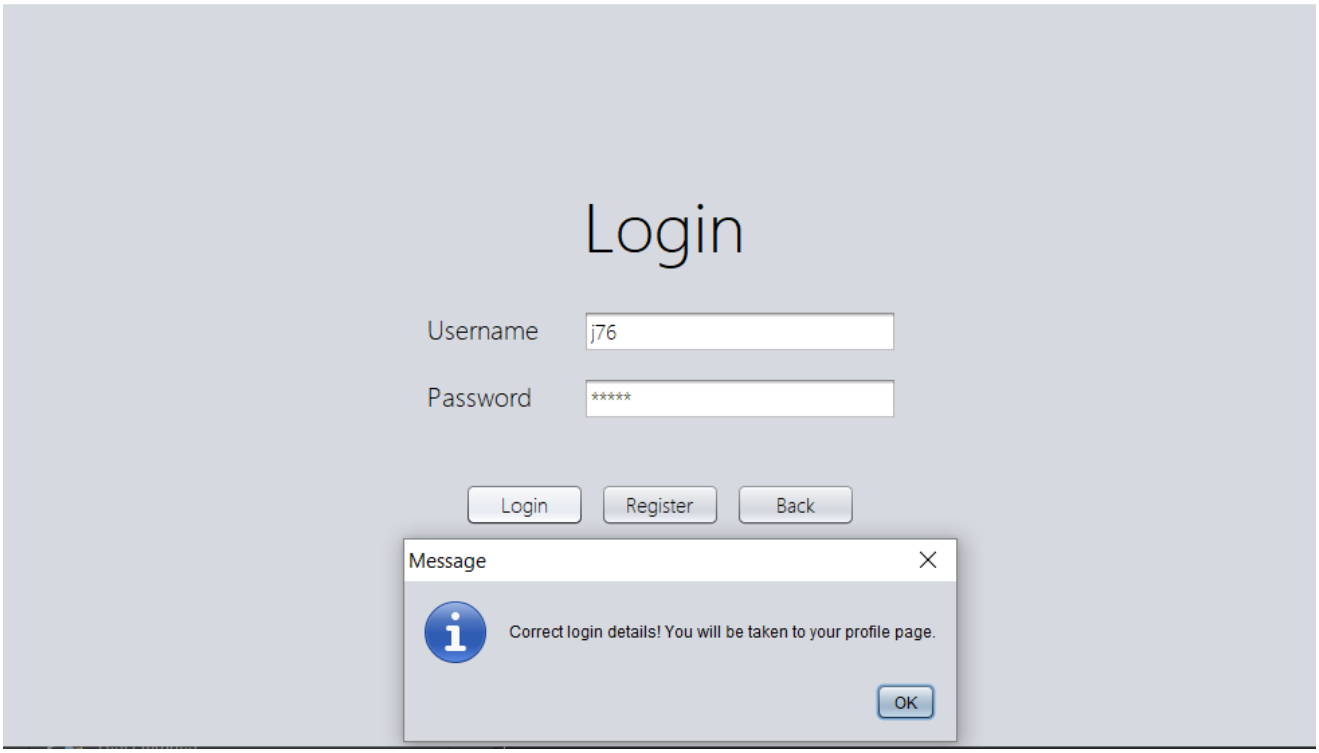

Figure 51 - Screenshot of a successful user login message.

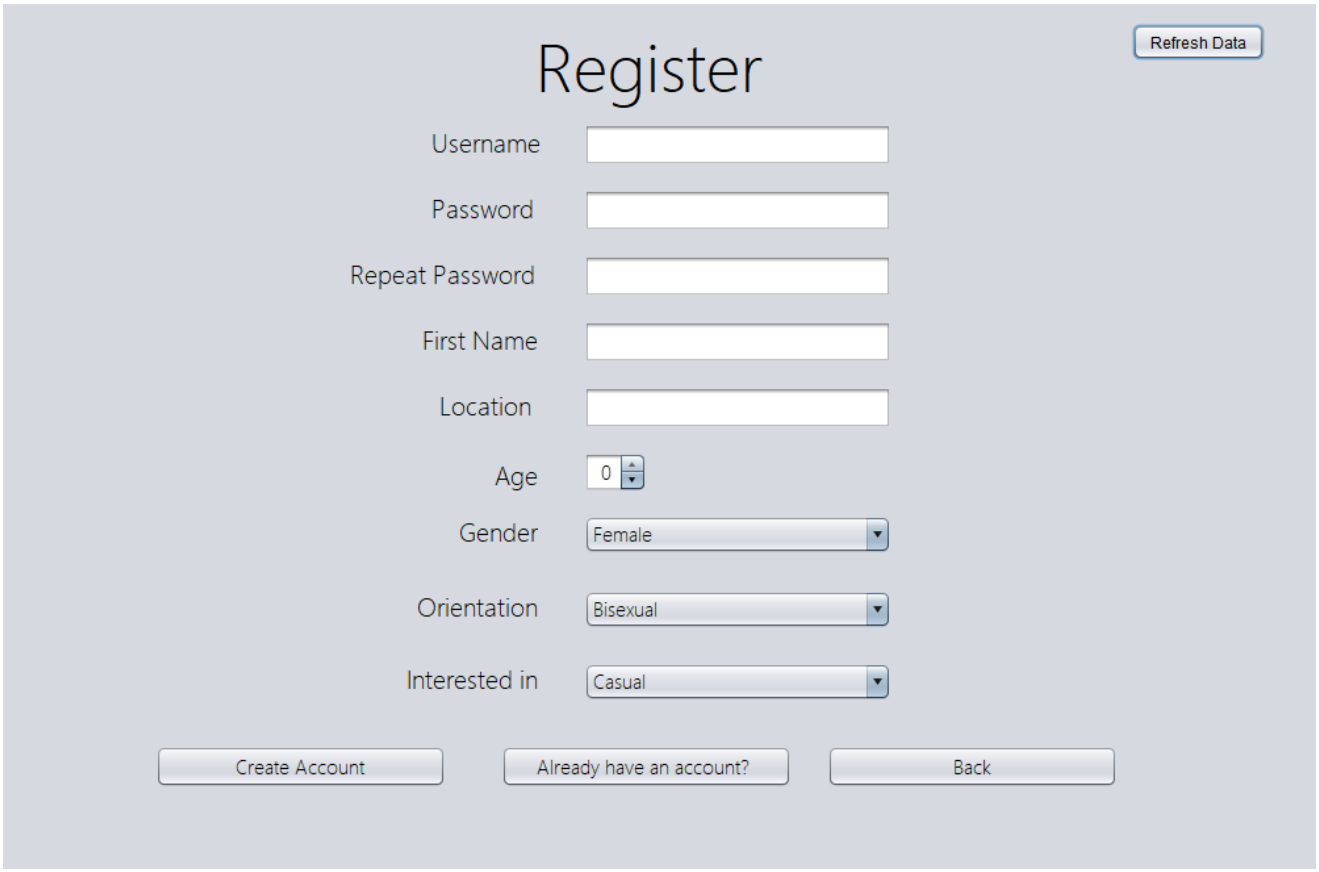

Figure 52 - Screenshot of the TechDating registers page.

Figure 53 - Screenshot of the profile page on the TechDating system.

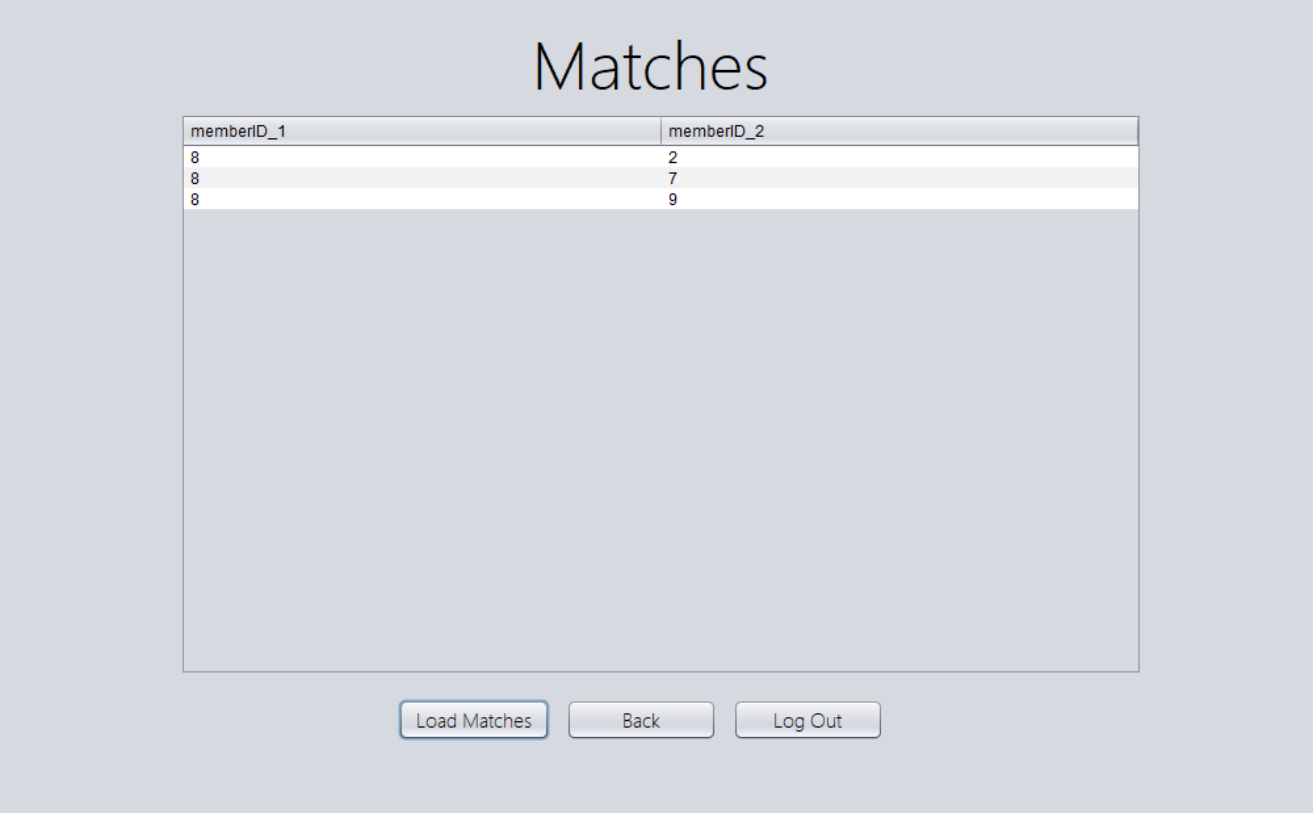

Figure 54 - Screenshot showing the user matches on the TechDating system.

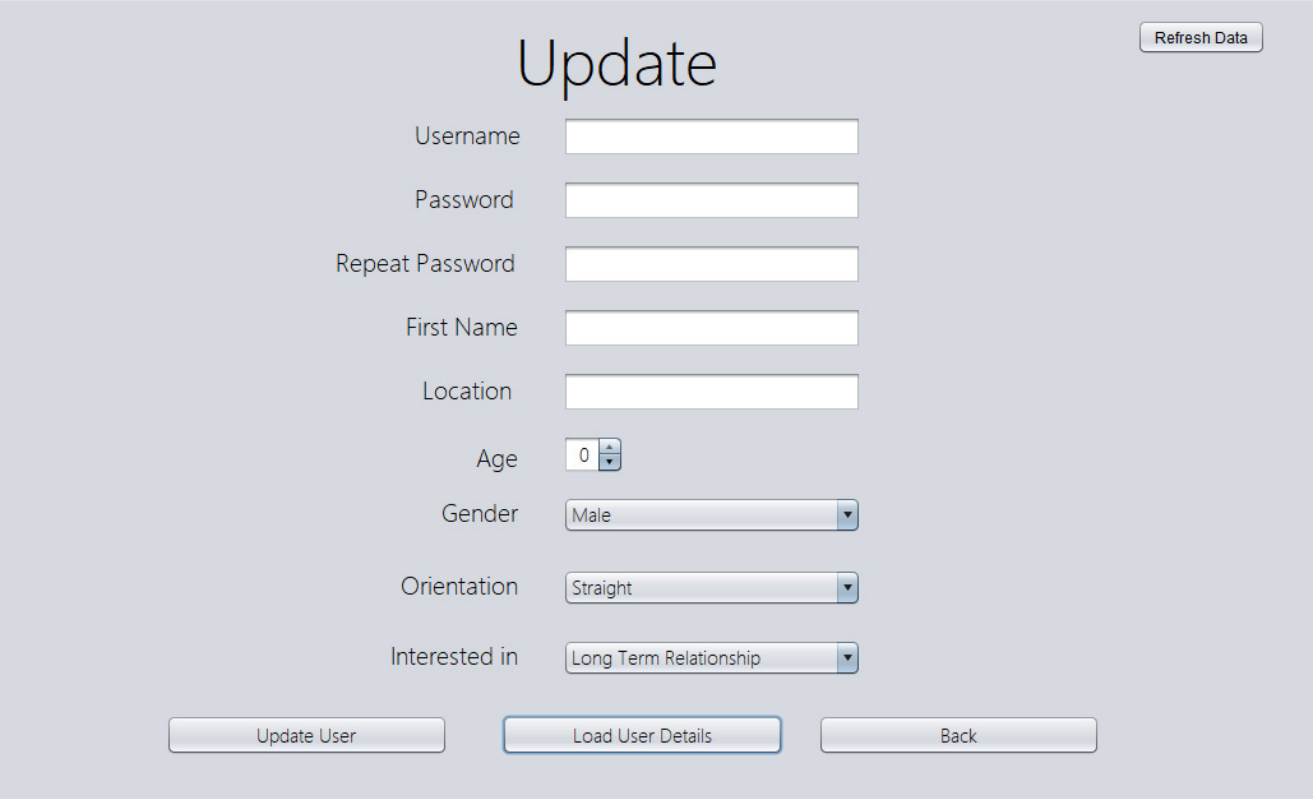

Figure 55 - Screenshot of the update users page on the TechDating system.

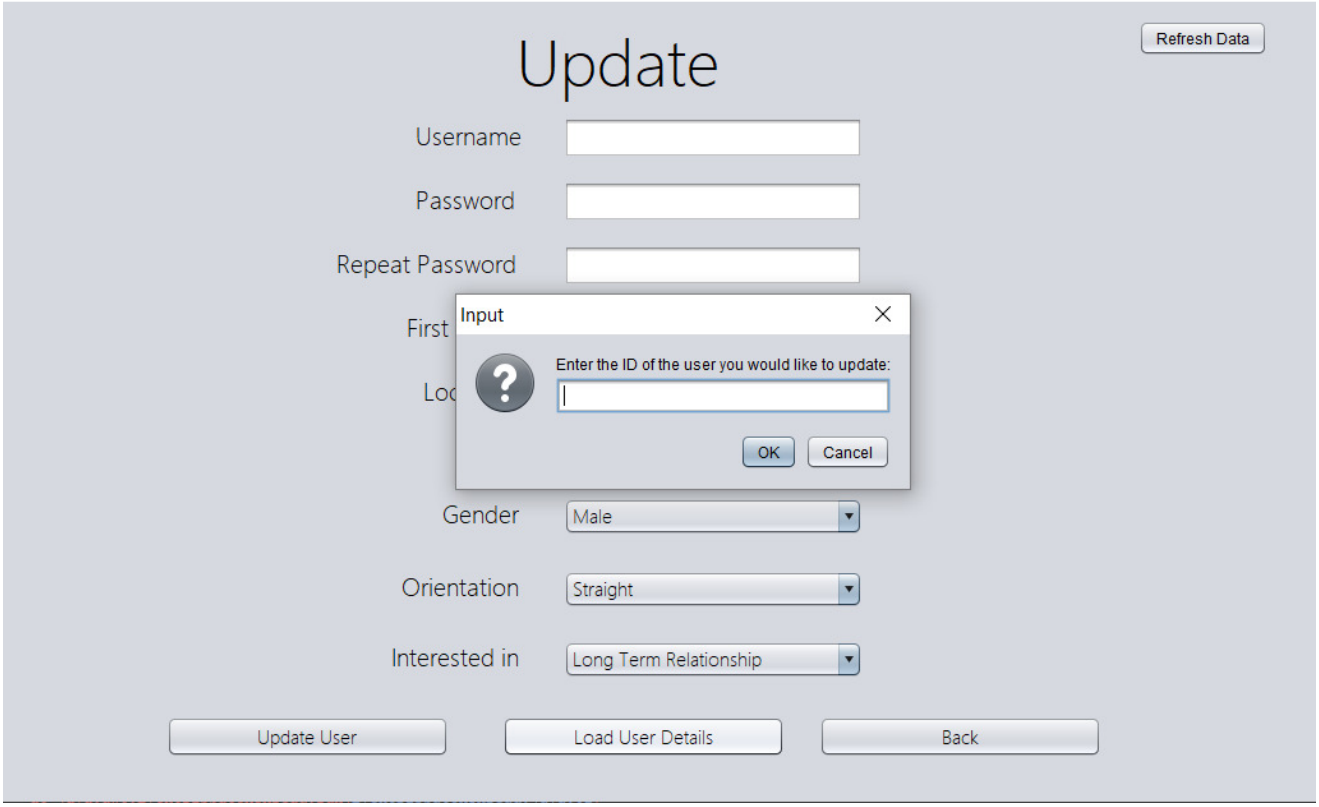

Figure 56 - Screenshot of the update users page when trying to load user information.

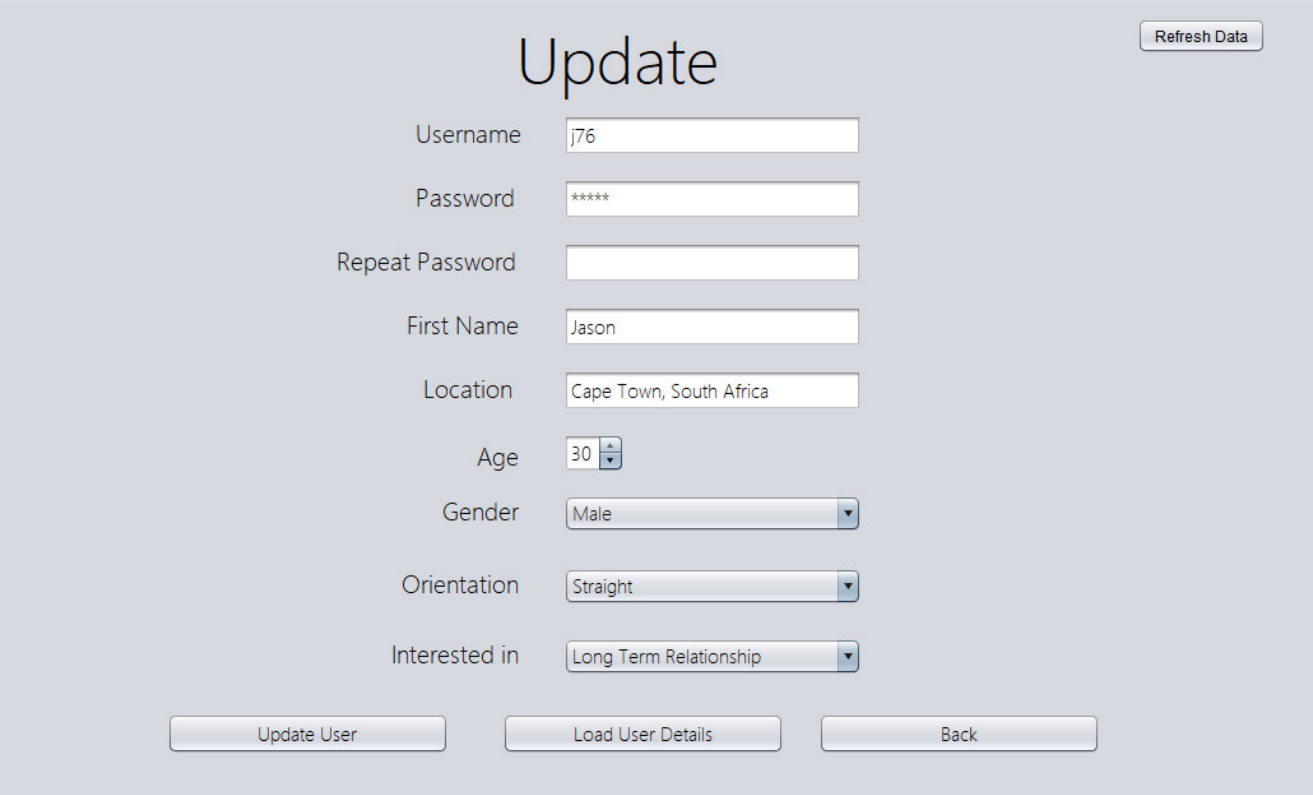

Figure 57 - Screenshot of the update users page with data loaded from the database.

CODE EXTRACTS

CONNECT TO THE DATABASE AND CLOSE THE CONNECTION

```java
public class DBConnect {
    static String url = Setup.DB_URL;
    static String username = Setup.DB_USERNAME;
    static String password = Setup.DB_PASSWORD;
    static String driver = Setup.DB_DRIVER;

    public static Connection getConnection() throws ClassNotFoundException, SQLException{
        Class.forName(driver);
        Connection con = null;
        con = DriverManager.getConnection(url, username, password);
        return con;
    }

    //close connection
    public static void close(Connection con, PreparedStatement pstmt, ResultSet rs) {
        try {
            if (rs != null) {
                rs.close();
            }
            if (pstmt != null) {
                pstmt.close();
            }
            if (con != null) {
                con.close();
            }
        }
        catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Figure 58 - Java code showing how to connect and close a database connection.

INSERT A USER INTO THE DATABASE

```java
try {
    conn = DBConnect.getConnection();
    Statement s = conn.createStatement();

    String insertMember = "INSERT INTO `techdatingdb`.`member`\n"
            + "( `firstname`, `age`, `location`, `gender`, `orientation`, `relationshipinterest`)\n"
            + "VALUES\n"
            + "('" + firstname + "',\n"
            + "'" + age + "',\n"
            + "'" + location + "',\n"
            + "'" + toIDGender(gender) + "',\n"
            + "'" + toIDOrient(orientation) + "','"
            + toIDInterest(interest) + "');";

    String insertLogin = "INSERT INTO `techdatingdb`.`login`\n"
            + "(`memberID`,\n"
            + "`username`,\n"
            + "`password`)\n"
            + "VALUES\n"
            + "((last_insert_id()),\n"
            + "'" + username + "',\n"
            + "'" + password + "');";

    JOptionPane.showMessageDialog(null, "Registration successful");

    s.execute(insertMember);
    s.execute(insertLogin);
    this.dispose();
    Profile profileFrame = new Profile();
    profileFrame.setVisible(true);
    SharedData.setUsername(username);

    conn.close();
    s.close();
} catch (ClassNotFoundException ex) {
    Logger.getLogger(Register.class.getName()).log(Level.SEVERE, null, ex);
    ex.printStackTrace();
} catch (SQLException ex) {
    Logger.getLogger(Register.class.getName()).log(Level.SEVERE, null, ex);
    ex.printStackTrace();
}
```

Figure 59 - Java code showing how to add a user to the database.

```
Connection conn = null;
PreparedStatement ps = null;
ResultSet rs = null;

try {
    //establish connection
    conn = DBConnect.getConnection();

    //query to retrieve all information from database
    ps = conn.prepareStatement("SELECT * FROM member");
    rs = ps.executeQuery();

    //create table model
    DefaultTableModel tableModel = new DefaultTableModel();
    //get meta data
    ResultSetMetaData metaData = rs.getMetaData();
    //get column names
    int columnCount = metaData.getColumnCount();
    for (int i = 1; i <= columnCount; i++) {
        tableModel.addColumn(metaData.getColumnLabel(i));
    }

    Object[] row = new Object[columnCount];

    while (rs.next()) {
        for (int i = 0; i < columnCount; i++) {
            row[i] = rs.getObject(i + 1);
        }
        tableModel.addRow(row);
    }

    //apply table model to member table
    memberTable.setModel(tableModel);

    //close all connections
    rs.close();
    ps.close();
    conn.close();

} catch (ClassNotFoundException | SQLException e) {
    DBConnect.close(conn, ps, rs);
    e.printStackTrace();
}
```

Figure 60 - Java code showing how to populate a table from the database.

POPULATE TEXTFIELDS FROM THE DATABASE

```
Connection conn = null;
PreparedStatement ps = null;
ResultSet rs = null;

String memberID = JOptionPane.showInputDialog(null, "Enter the ID of the user you would like to update:");

int id = Integer.parseInt(memberID);
SharedData.setMemberID(memberID);
try {
    conn = DBConnect.getConnection();
    ps = conn.prepareStatement("SELECT * FROM\n"
            + "`techdatingdb`.`member`\n"
            + "INNER JOIN `techdatingdb`.`login` on login.memberID = member.memberID\n"
            + "where member.memberID = '" + id + "';");
    rs = ps.executeQuery();

    while (rs.next()) {
        txtUsername.setText(rs.getString("username"));
        txtPassword.setText(rs.getString("password"));
        txtFirstName.setText(rs.getString("firstname"));
        txtLocation.setText(rs.getString("location"));
        spinAge.setValue(rs.getInt("age"));
        cmbGender.setSelectedItem(rs.getString("gender"));
        cmbOrientation.setSelectedItem(rs.getString("orientation"));
        cmbInterest.setSelectedItem(rs.getString("relationshipinterest"));

    }
} catch (ClassNotFoundException | SQLException e) {
    DBConnect.close(conn, ps, rs);
    e.printStackTrace();
}
```

Figure 61 - Java code to populate text fields from the database.

UPDATE A RECORD IN THE DATABASE

```
Connection conn = null;

int ID = Integer.parseInt(SharedData.getMemberID());
if (!password.equals(passwordRepeat)) {
    JOptionPane.showMessageDialog(null, "Passwords do not match");
} else //insert data into the database
{
    try {
        conn = DBConnect.getConnection();
        Statement s = conn.createStatement();

        String insertMember = "UPDATE `techdatingdb`.`member`\n"
                + "SET `firstname` = '" + firstname
                + "', `age` = '" + age
                + "', `location` = '" + location
                + "', `gender` = '" + toIDGender(gender)
                + "', `orientation` = '" + toIDOrient(orientation)
                + "', `relationshipinterest` = '" + toIDInterest(interest)
                + "' WHERE memberID =" + ID
                + ";";

        String insertLogin = "UPDATE `techdatingdb`.`login`\n"
                + "SET `password` = '" + password
                + "' WHERE memberID =" + ID
                + ";";

        JOptionPane.showMessageDialog(null, "Update successful");

        s.execute(insertMember);
        s.execute(insertLogin);

        conn.close();
        s.close();
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(Register.class.getName()).log(Level.SEVERE, null, ex);
        ex.printStackTrace();
    } catch (SQLException ex) {
        Logger.getLogger(Register.class.getName()).log(Level.SEVERE, null, ex);
        ex.printStackTrace();
    }
}
```

Figure 62 - Java code to update records in the database.

DELETE A RECORD IN THE DATABASE

```
Connection conn = null;
PreparedStatement ps = null;
ResultSet rs = null;

String memberID = JOptionPane.showInputDialog(null, "Enter the ID of the user you would like to delete:");

int id = Integer.parseInt(memberID);

//delete user from login and member tables
try {
    conn = DBConnect.getConnection();
    ps = conn.prepareStatement("set foreign_key_checks = 0;\n"
            + "DELETE FROM `techdatingdb`.`member`, `techdatingdb`.`login`\n"
            + "USING `techdatingdb`.`member`\n"
            + "INNER JOIN `techdatingdb`.`login` WHERE `login`.`memberID` = '" + id + "' AND login.memberID = member.memberID;\n"
            + "set foreign_key_checks = 1;");

    rs = ps.executeQuery();

} catch (ClassNotFoundException | SQLException e) {
    DBConnect.close(conn, ps, rs);
    e.printStackTrace();
}
```

Figure 63 - Java code to delete a user from the database.

CALL A STORED PROCEDURE

```
try {

    //establish connection
    conn = DBConnect.getConnection();

    //query to call the get profile procedure
    ps = conn.prepareStatement("CALL get_profile("+ id+ ");");
    rs = ps.executeQuery();

    //create table model
    DefaultTableModel tableModel = new DefaultTableModel();
    //get meta data
    ResultSetMetaData metaData = rs.getMetaData();
    //get column names
    int columnCount = metaData.getColumnCount();
    for (int i = 1; i <= columnCount; i++) {
        tableModel.addColumn(metaData.getColumnLabel(i));
    }

    Object[] row = new Object[columnCount];

    while (rs.next()) {
        for (int i = 0; i < columnCount; i++) {
            row[i] = rs.getObject(i + 1);
        }
        tableModel.addRow(row);
    }

    //apply table model to tblProfile
    tblProfile.setModel(tableModel);

    //close all connections
    rs.close();
    ps.close();
    conn.close();

} catch (ClassNotFoundException | SQLException e) {
    DBConnect.close(conn, ps, rs);
    e.printStackTrace();
}
```

Figure 64 - Java code to call a stored procedure.

CONCLUSION

The TechDating System was required to meet the following requirements:

Requirement specification	Was the requirement met in the TechDating System?	Remarks
ERD modelling	Yes	The TechDating System has been modelled with conceptual, logical and physical designs.
Data normalization - 3rd level	Yes	The TechDating System has been normalised to 3NF.
Entity integrity maintained	Yes	All primary keys in the TechDating System are unique and not null.
Referential integrity maintained	Yes	All foreign keys in the TechDating System are valid and reference a valid primary key.
Cohesive modelling of a conceptual model to a logical model	Yes	The conceptual model has been cohesively modelled to the logical model, with consistent naming conventions and relationships throughout all the models.
Entity types identified and understood	Yes	The entity types are identified, and their attributes and properties defined.
Relevant indexes identified and implemented	Yes	Indexes were created and implemented within the database to improve search time on queries.
Relevant views identified and implemented	Yes	The stored procedure get_profile acted as a view in this database, whereby it created a virtual table that users can view but not modify.
Cohesive integration between application or website and database	Yes	The database connected to the application and queries stored procedures and triggered can be executed through the application.
Database runs	Yes	The database runs and is able to execute all stored procedures, triggers, CRUD operations and connect to the application.
The database should have the following:		
5 entities	Yes	7 entities were created: Member, Login, Gender, Orientation, Relationship Interest, Match, and Messages.
1 trigger	Yes	1 trigger was created: Check age.
2 stored procedures	Yes	2 stored procedures were created: Create a match and get a profile
2 indexes	Yes	4 indexes were created: Gender, orientation, relationship interest and username.
2 users	Yes	2 users were created: TechDating member and administrator.

Figure 65 - Requirements for a fully functional database.

As shown in the table above, all requirements for a fully functional database were met - thus the TechDating System and Database are successful in implementation.

QUESTION 1.2.A.

A Role in MySQL is a named collection of user's who have the same privileges within a database.

The database will have two users of the database, an admin and a user, thus these will each have their own roles. Shown below is a screenshot displaying the two users and various roles of the TechDating Database (MySQL, 2018):

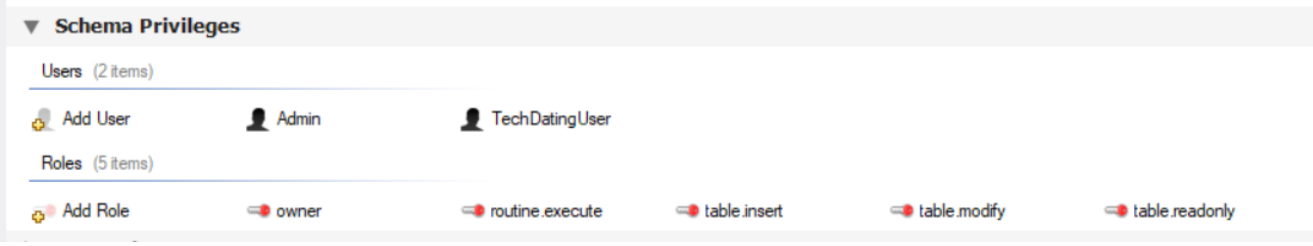

Figure 66 - Users and Roles of TechDating Database.

The admin user has the following roles:

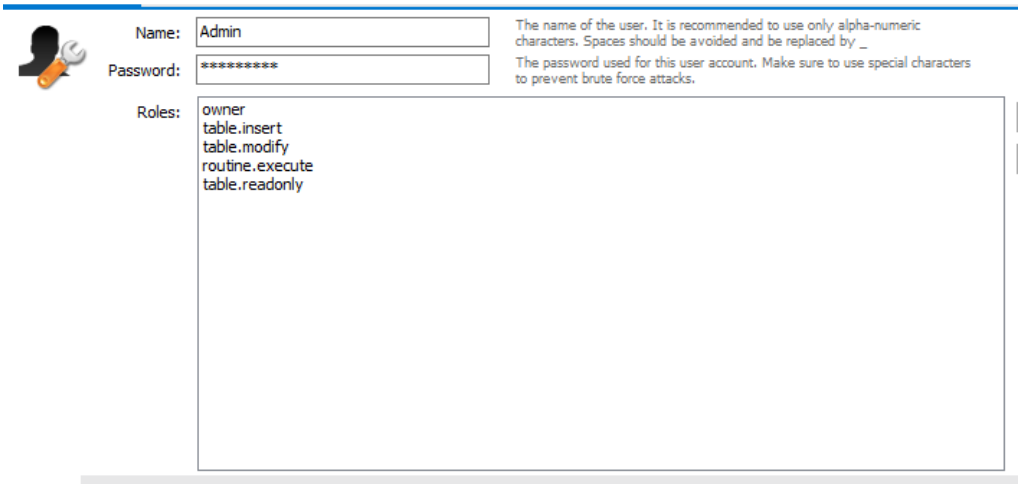

Figure 67 - Admin roles for the TechDating Database

The TechDating user has the following roles:

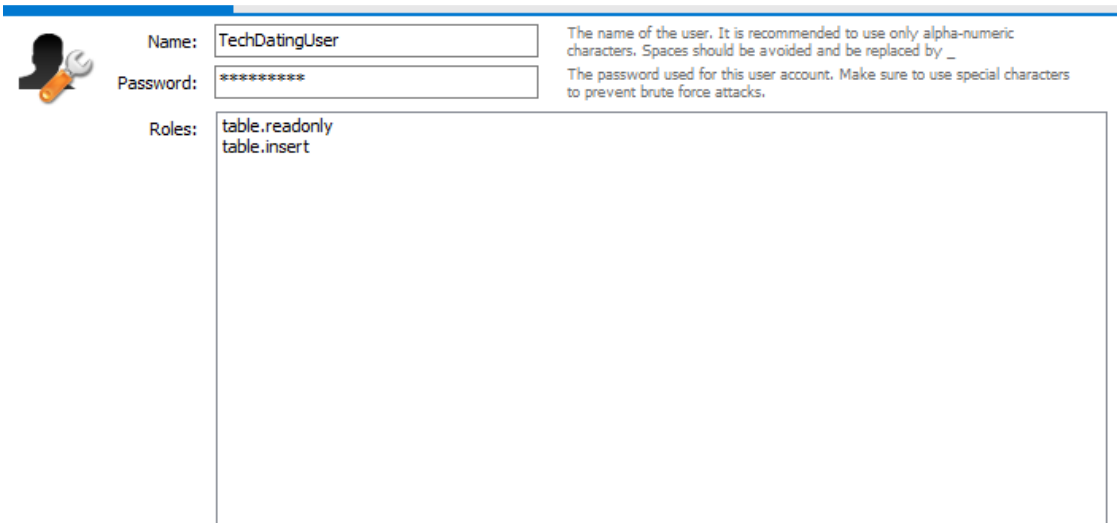

Figure 68 - User roles for the TechDating Database.

The database roles assigned to the users have the following privileges:

Owner:

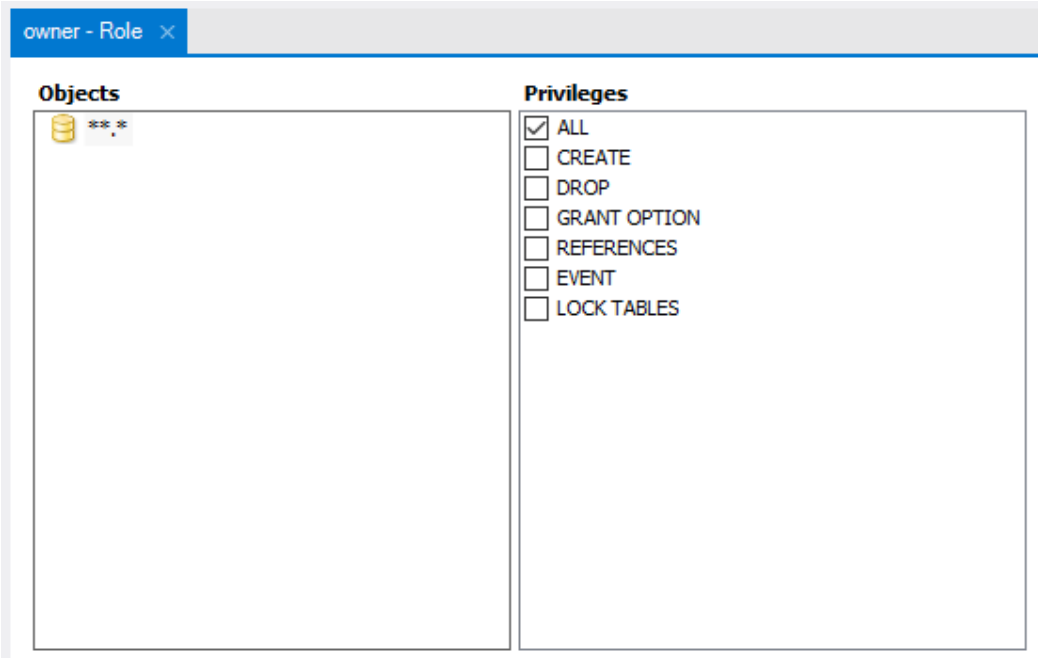

Figure 69 - Privileges for the Owner role.

Routine Execute:

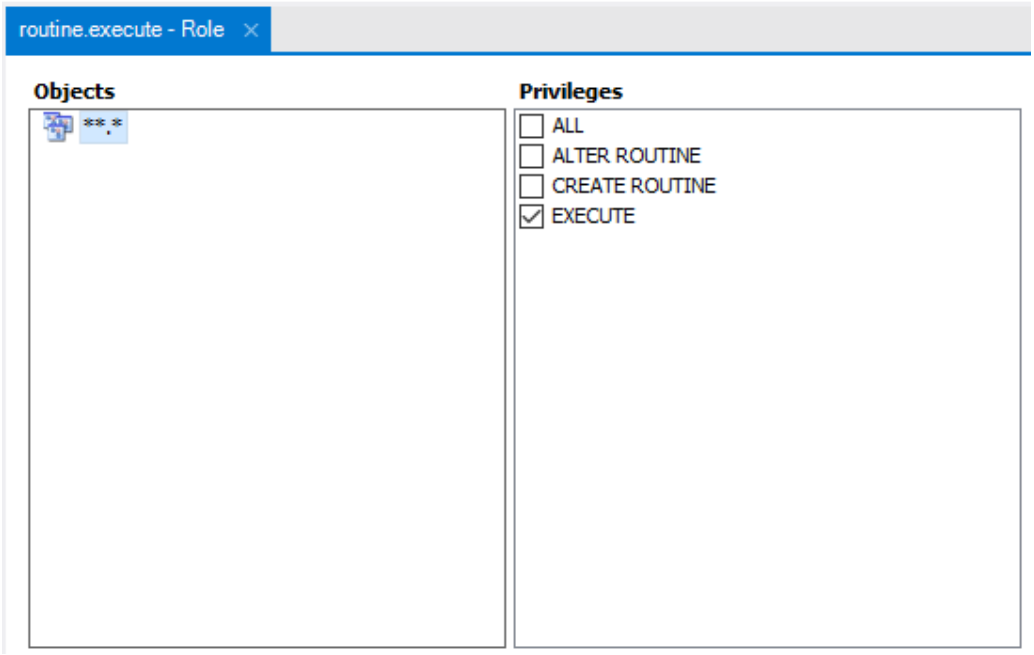

Figure 70 - Privileges for the Routine Execute Role.

Table Insert:

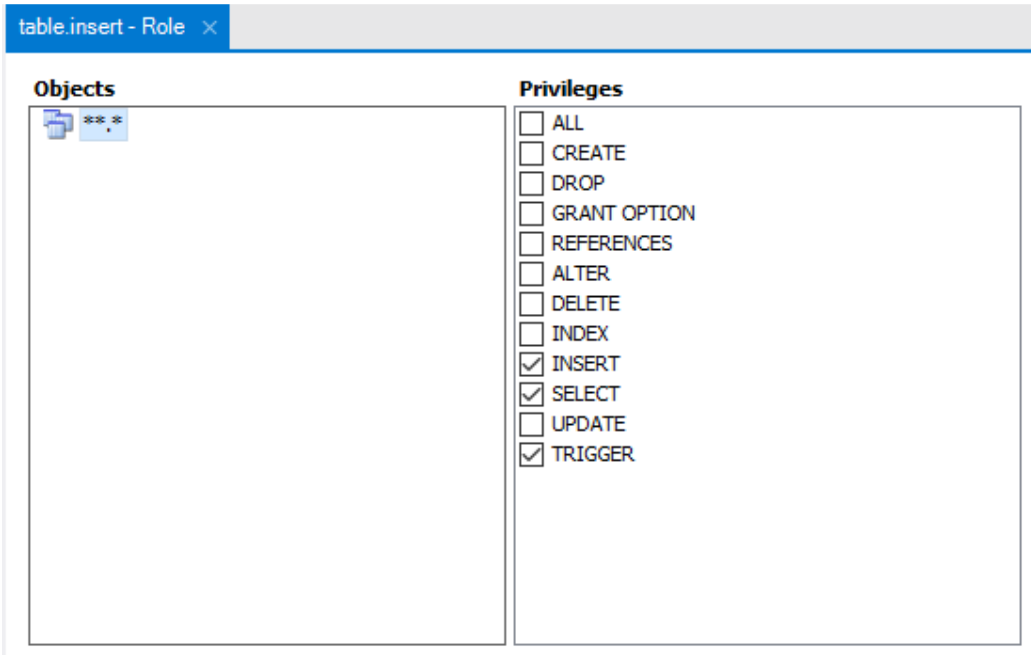

Figure 71 - Privileges for the Table Insert Role.

Table Modify:

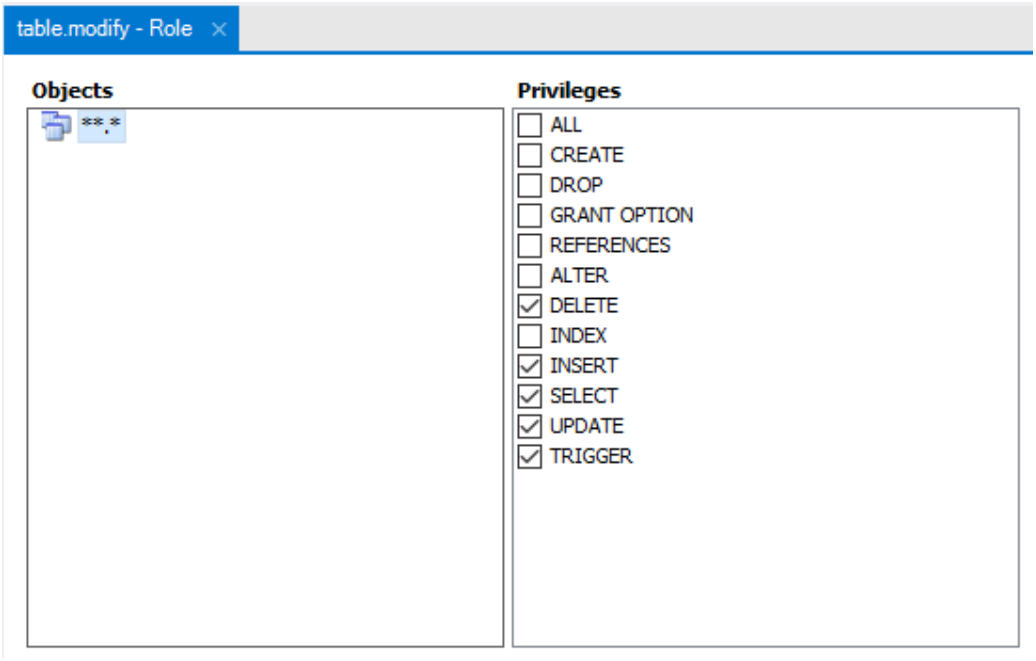

Figure 72 - Privileges for the Table Modify Role.

Table Read Only:

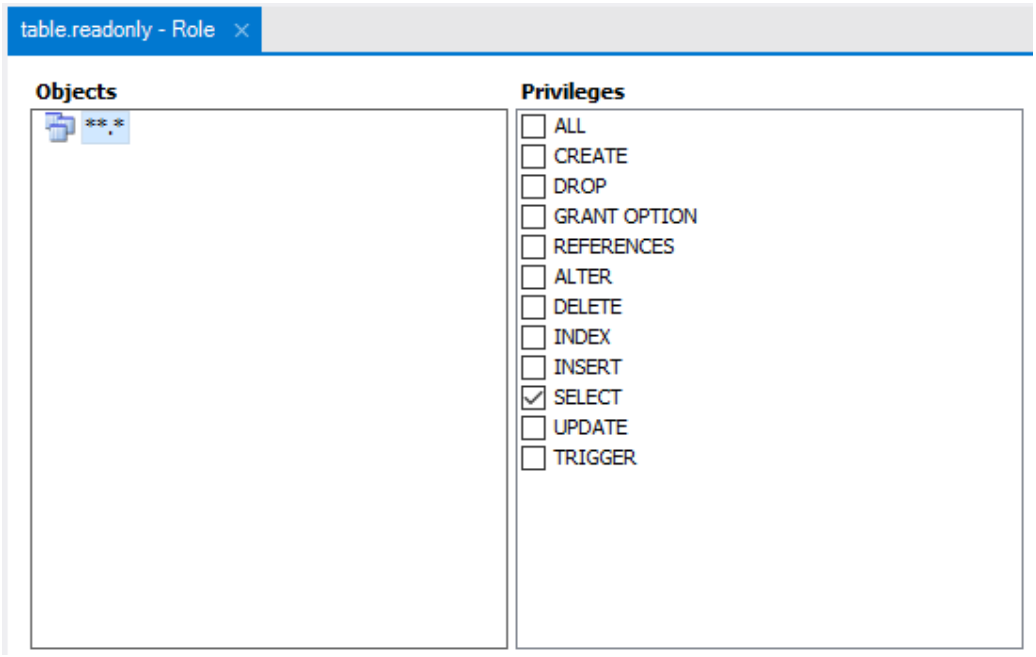

Figure 73 - Privileges for the Table Read Only Role.

The following table outlines the privileges for two users of the system according to their roles:

	ADMIN USER	TECHDATING USER
USER ROLES	owner, table.insert, table.modify, routine.execute, table.readonly	table.readonly, table.insert
PRIVILEGE	ADMIN USER	TECHDATING USER
Create	Yes	
Drop	Yes	
Grant option	Yes	
References	Yes	
Event	Yes	
Lock tables	Yes	
Execute	Yes	
Insert	Yes	Yes
Select	Yes	Yes
Trigger	Yes	
Delete	Yes	
Update	Yes	Yes

Table 10 - Privileges for the users of the TechDating System.

As shown above, the two users of the database - Admin and TechDating User - have been assigned roles according to what privileges they need to use the system, with a user only needing to register (insert), view data (select), and update their personal details and send messages (update), while the admin needs to have full access to the system and be able to use all functionalities.

QUESTION 1.2.B.

Privileges are granted to users in order for them to accomplish tasks they need to do to use the system, a privilege is defined by (Oracle, 2002) as the right to execute a particular type of SQL statement or access another users object. Global privileges provide users with the rights to perform actions on any schema objects, such as allowing a user to modify any database tables within the system - it is recommended that global privileges only be granted to database administrators and developers for security purposes (Oracle, 2002). Schema privileges provide users with the right to perform actions on a specific schema object, such as a table or procedure within a database - these privileges should be granted to end users in order for them to perform their required tasks (Oracle, 2002).

MySQL has the following types of privileges that can be granted to users:

- Administrative privileges
    - These enable users to manage the operation of a MySQL Server;
    - They are not specific to a specific database, therefore referred to as global privileges (MySQL, 2018).
- Database privileges
    - These apply to a database and all objects within the database (MySQL, 2018).
- Database object privileges
    - These apply to specific database objects such as tables, indexes, views or stored procedures within the database (MySQL, 2018).

The following screenshot shows all users for the MySQL Server:

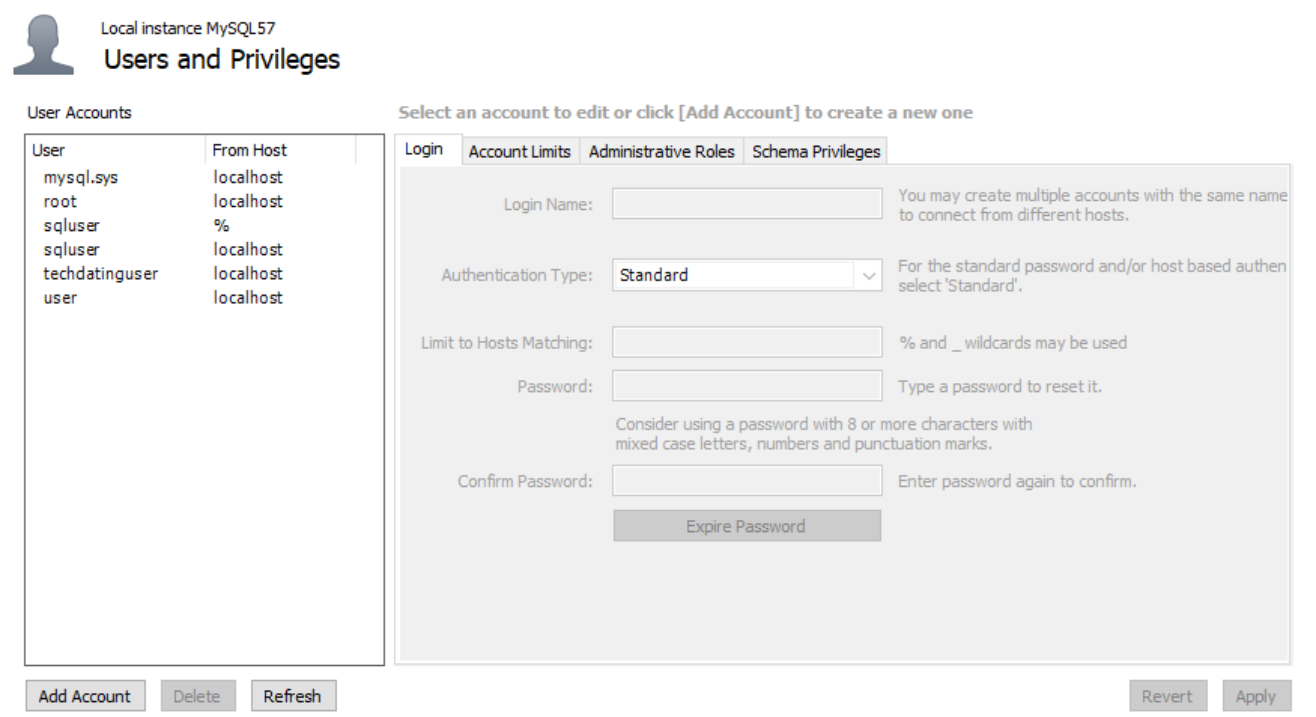

Figure 74 - Users and Privileges for the MySQL Server.

The two users relevant to the TechDating Database are *root@localhost* for the admin user, and *techdatinguser@localhost* for the user of the TechDating System. A new account needed to be made for *techdatinguser@localhost*.

The following screenshots outline all details for the *root@localhost* user and the relevant global and schema privileges for that user:

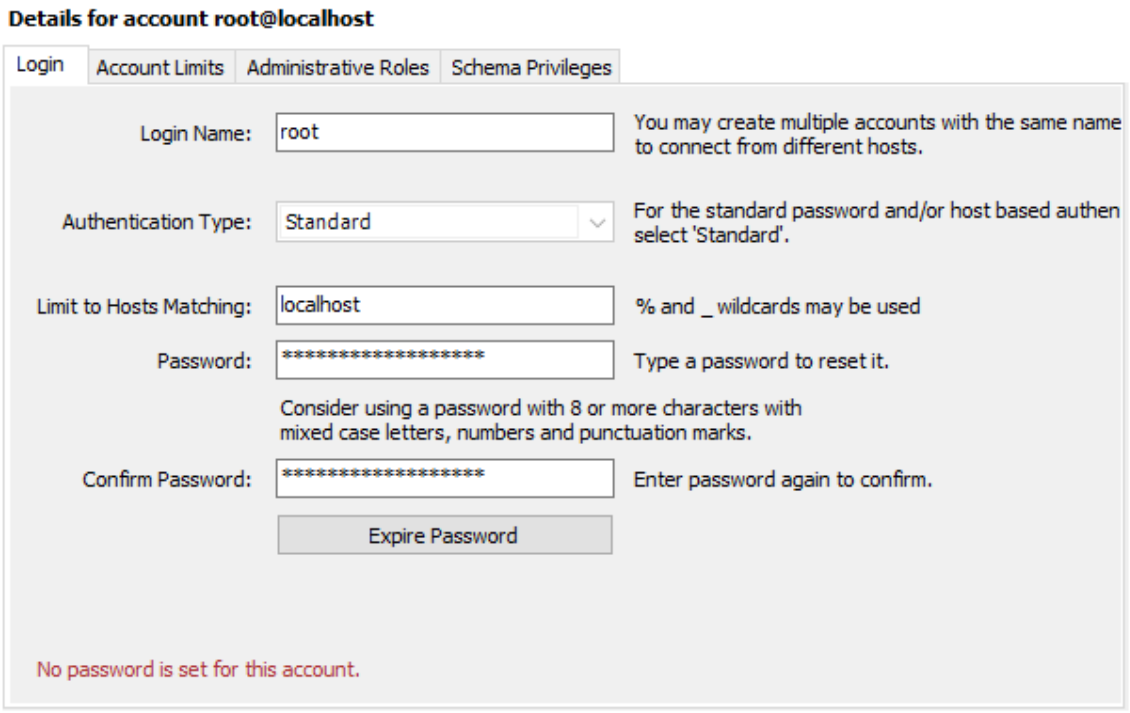

Figure 75 - Details for the account root@localhost.

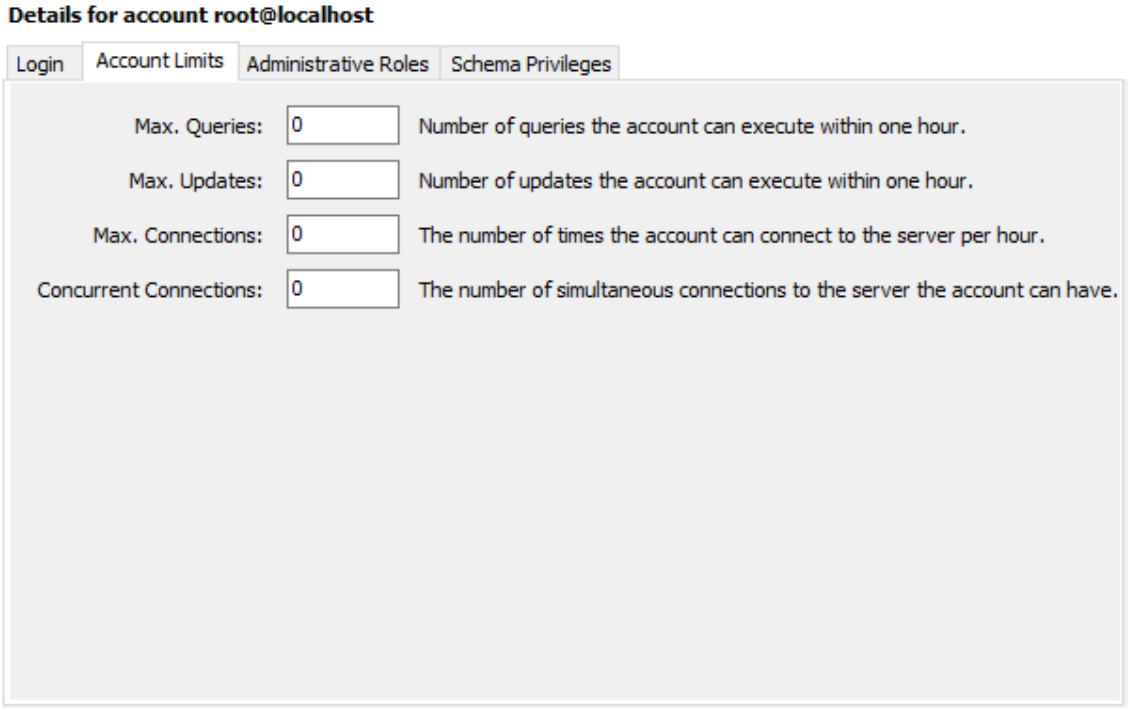

Figure 76 - Details for the account root@localhost.

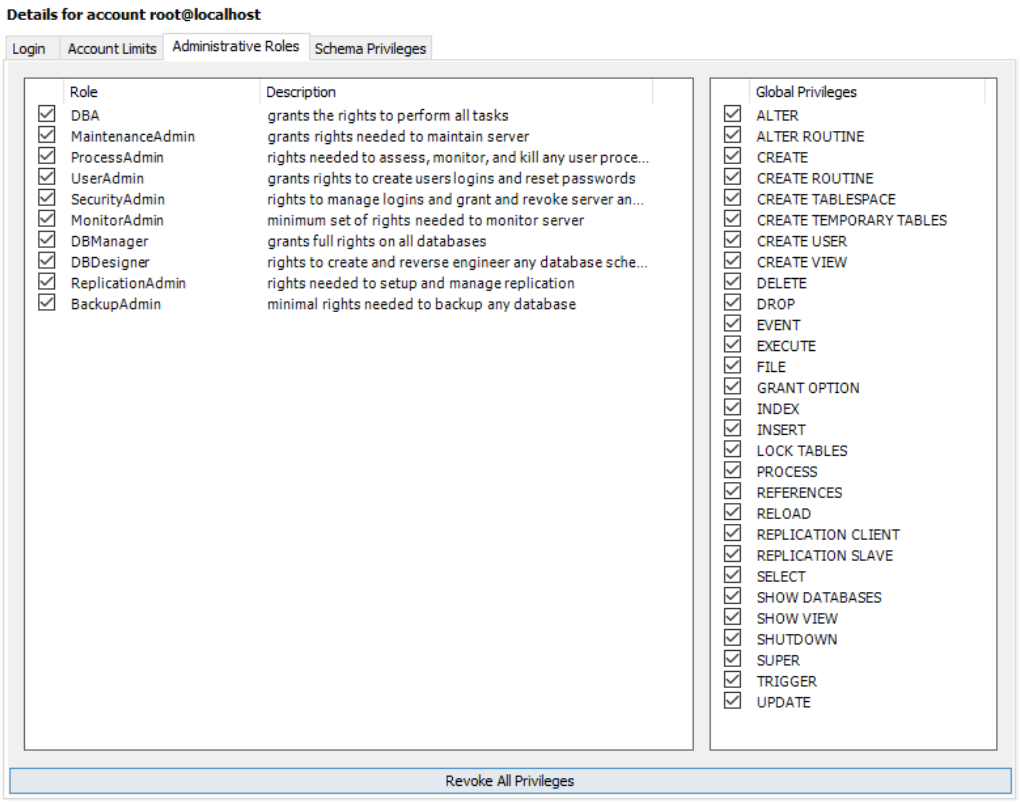

Figure 77 - Global privileges for the account root@localhost.

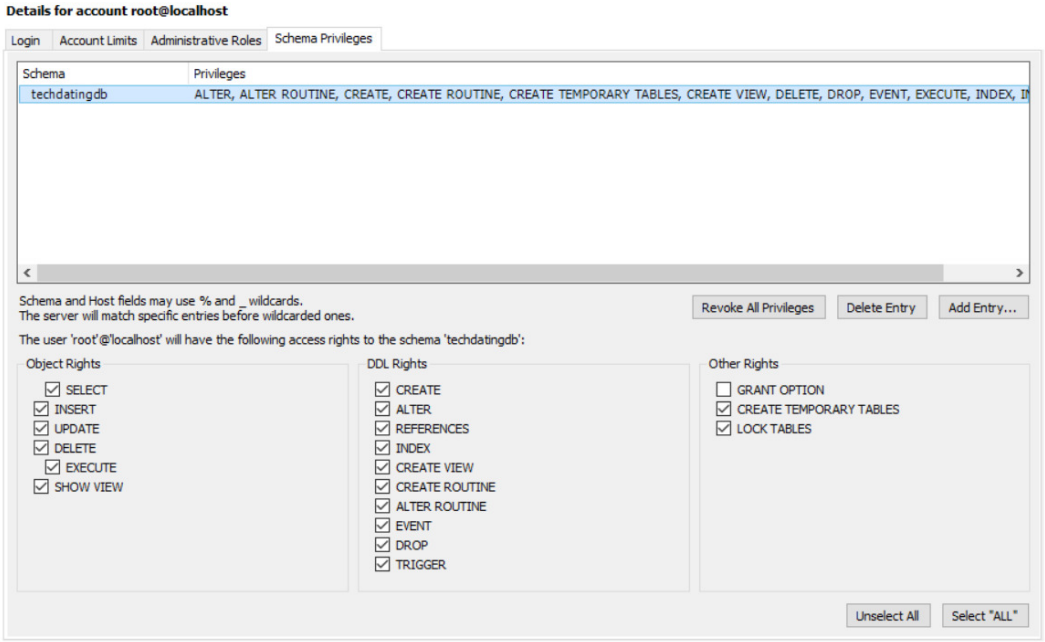

Figure 78 - Schema privileges for the account root@localhost.

The user *root@localhost* is the database administrator, thus holds all global privileges as well as all schema privileges for the TechDating Database, as shown in Figure 77 and Figure 78.

The following screenshots outline all details for the *techdatinguser@localhost* user and the relevant global and schema privileges for that user:

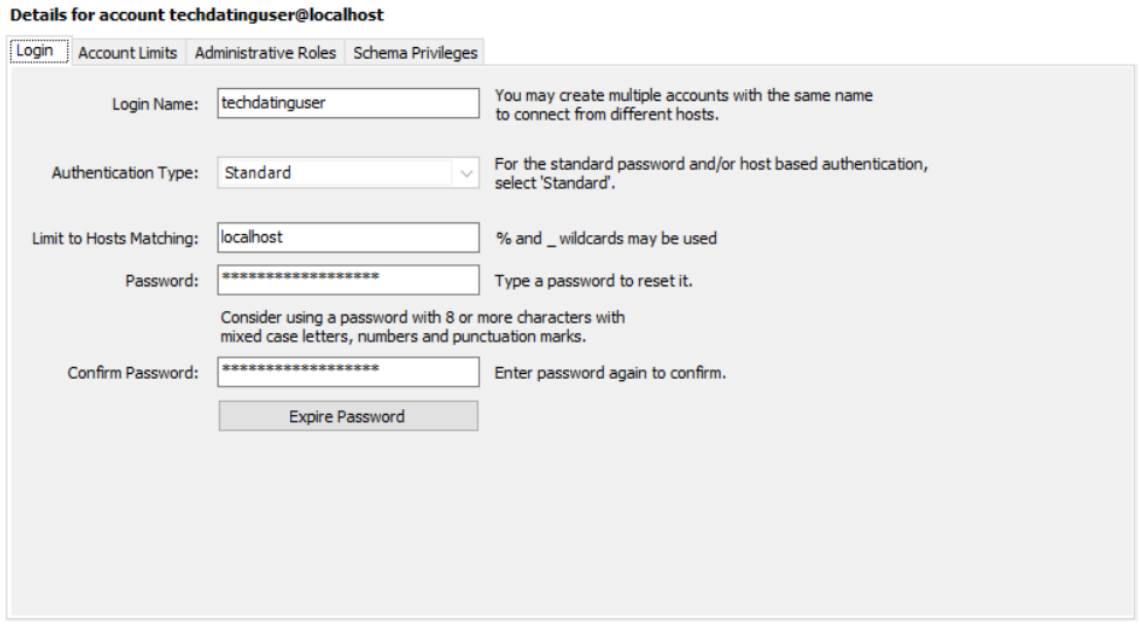

Figure 79 - Details for the account techdatinguser@localhost.

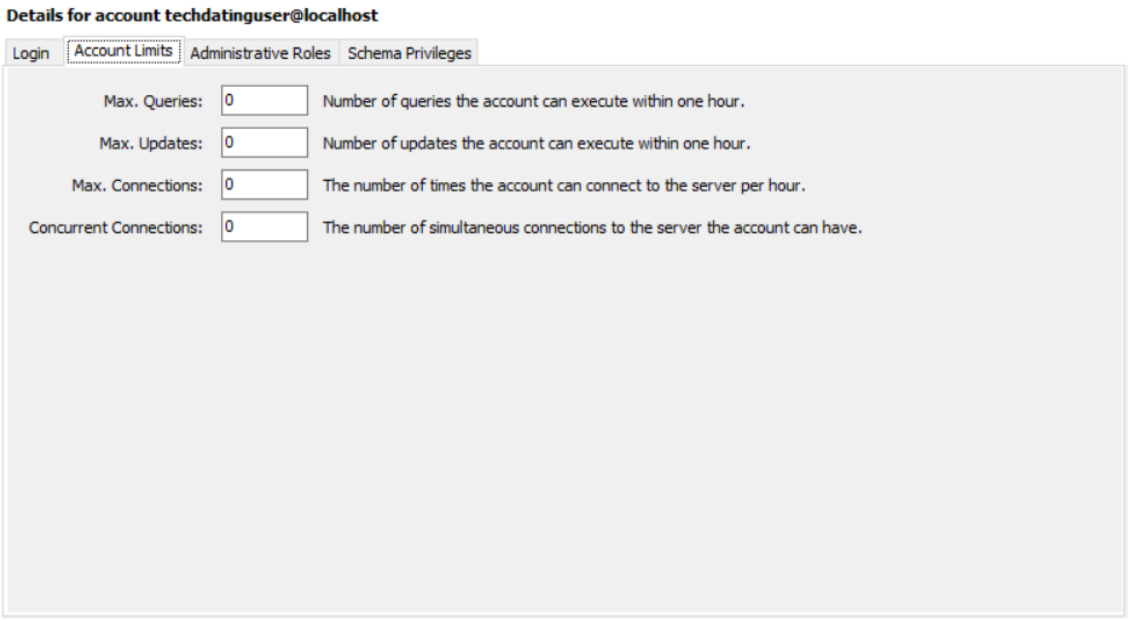

Figure 80 - Details for the account techdatinguser@localhost.

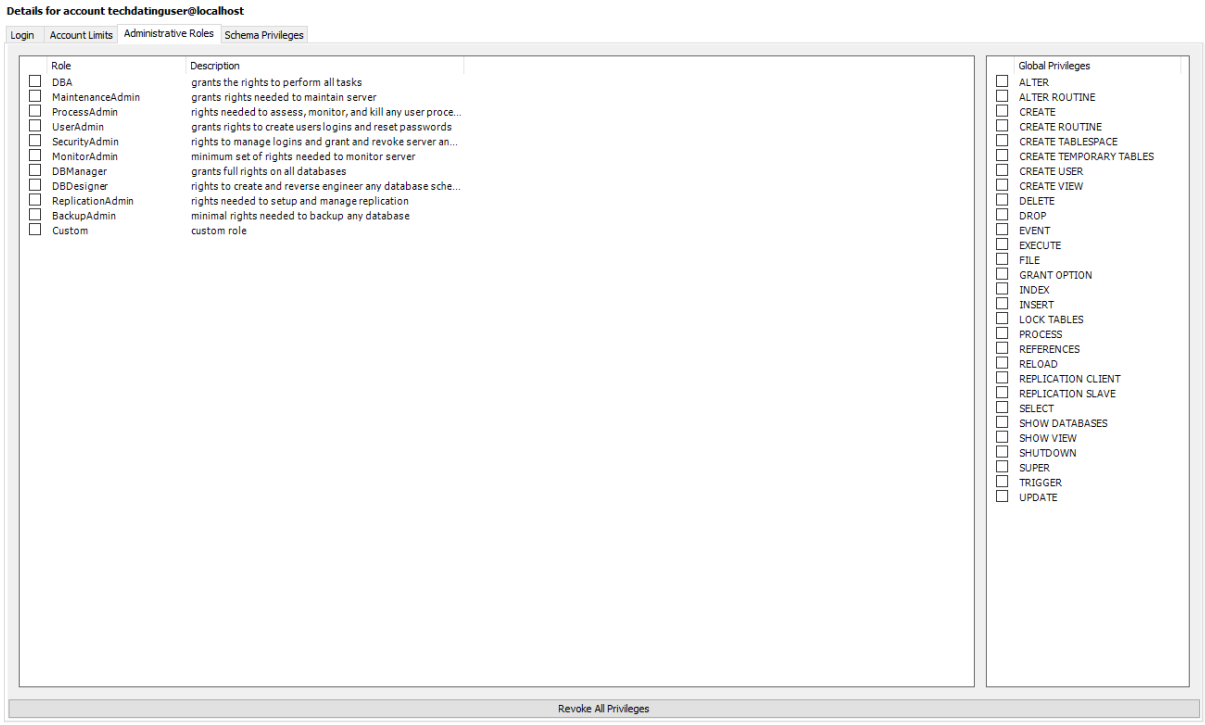

Figure 81 - Global privileges for the user techdatinguser@localhost.

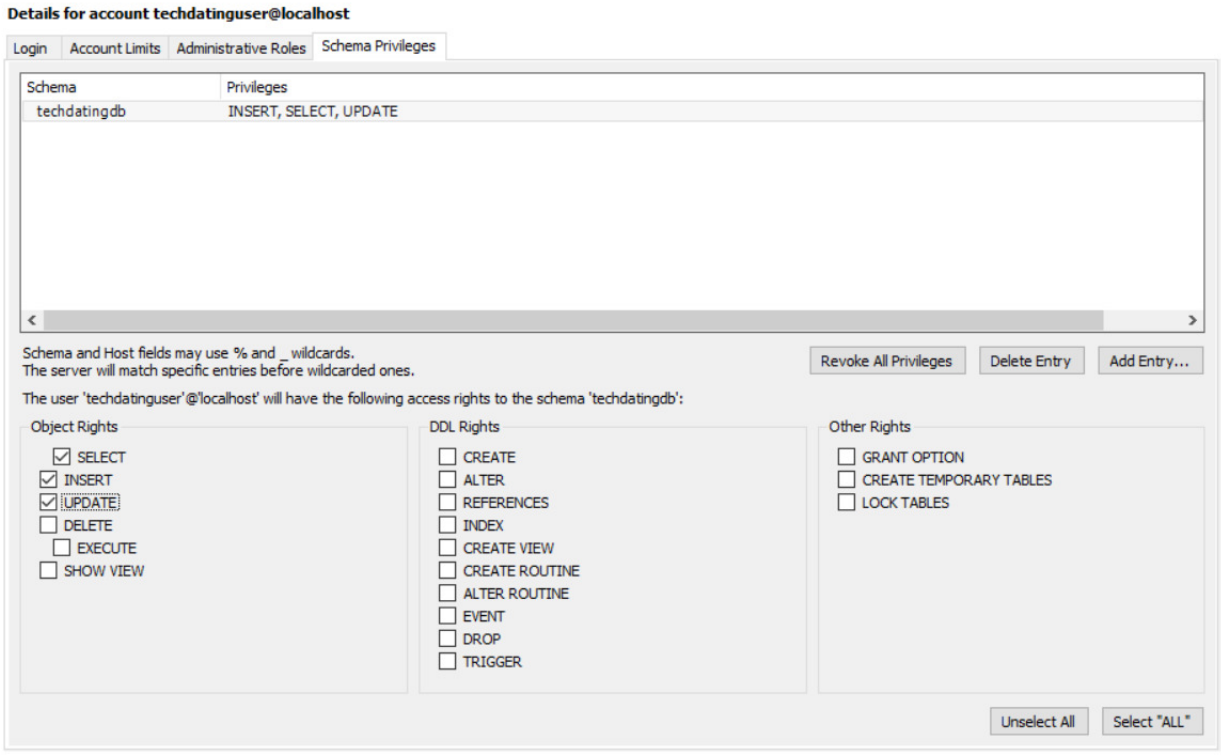

Figure 82 - Schema privileges for the user techdatinguser@localhost.

The user *techdatinguser@localhost* is the database user, thus they do not hold any global privileges as they only need to access and perform actions on the TechDating Database, they hold the required schema privileges for their role for the TechDating Database, that being to select, insert and update data on the tables within the database, as shown in Figure 81 and Figure 82.

QUESTION 2

QUESTION 2.1.

Database security is defined by (Connolly & Begg, 2015) as "the mechanisms that protect the database against intentional or accidental threats". Data is one of an organisation's most valuable assets, thus must be treated as such and appropriate security measures should be taken in order to protect it; security breaches affect not only the data stored within a database, but can affect parts of or the entire organisation's system - which in turn will affect the database - thus database security should take into consideration hardware, software, people, and data (Connolly & Begg, 2015). The need for security has become more prevalent within organisations as of recently due to the increased amounts of important organisational data being stored on databases - where the loss or unavailability of the data can be disastrous to the organisation (Connolly & Begg, 2015).

Database security should be implemented in order to minimise the risk of the aforementioned scenarios from occurring:

- Accidental loss;
- Loss of availability;
- Loss of privacy;
- Loss of confidentiality and/or secrecy;
- Theft and fraud;
- Loss of data integrity (Connolly & Begg, 2015) (Singh, 2011) (Burtescu, 2009).

Figure 2 below outlines examples of threats organisations face and their impact on the organisation:

THREAT	THEFT AND FRAUD	LOSS OF CONFIDENTIALITY	LOSS OF PRIVACY	LOSS OF INTEGRITY	LOSS OF AVAILABILITY
Using another person's means of access	✓	✓	✓		
Unauthorized amendment or copying of data	✓			✓	
Program alteration	✓			✓	✓
Inadequate policies and procedures that allow a mix of confidential and normal output	✓	✓	✓		
Wire tapping	✓	✓	✓		
Illegal entry by hacker	✓	✓	✓		
Blackmail	✓	✓	✓		
Creating "trapdoor" into system	✓	✓	✓		
Theft of data, programs, and equipment	✓	✓	✓		✓
Failure of security mechanisms, giving greater access than normal		✓	✓	✓	
Staff shortages or strikes				✓	✓
Inadequate staff training		✓	✓	✓	✓
Viewing and disclosing unauthorized data	✓	✓	✓		
Electronic interference and radiation				✓	✓
Data corruption owing to power loss or surge				✓	✓
Fire (electrical fault, lightning strike, arson), flood, bomb				✓	✓
Physical damage to equipment				✓	✓
Breaking cables or disconnection of cables				✓	✓
Introduction of viruses				✓	✓

Figure 83 - Examples of Threats. Source: (Connolly & Begg, 2015, p. 610).

Connolly & Begg (2015) define a threat as “any situation or event, whether intentional or accidental, that may adversely affect a system and consequently the organisation”. Threats can be caused by an event or situation that involves people, actions, or any circumstances that could harm organisations - in any way including tangible losses such as hardware, software, or data, and including intangible losses such as client confidence and organisational credibility (Connolly & Begg, 2015). All threats faced by an organisation should be viewed as a potential security breach, as they will have an impact on the business productivity and profits if they are successful (Connolly & Begg, 2015).

The following diagram outlines the possible threats to Bophelo Hospital patient data, which are elaborated further below:

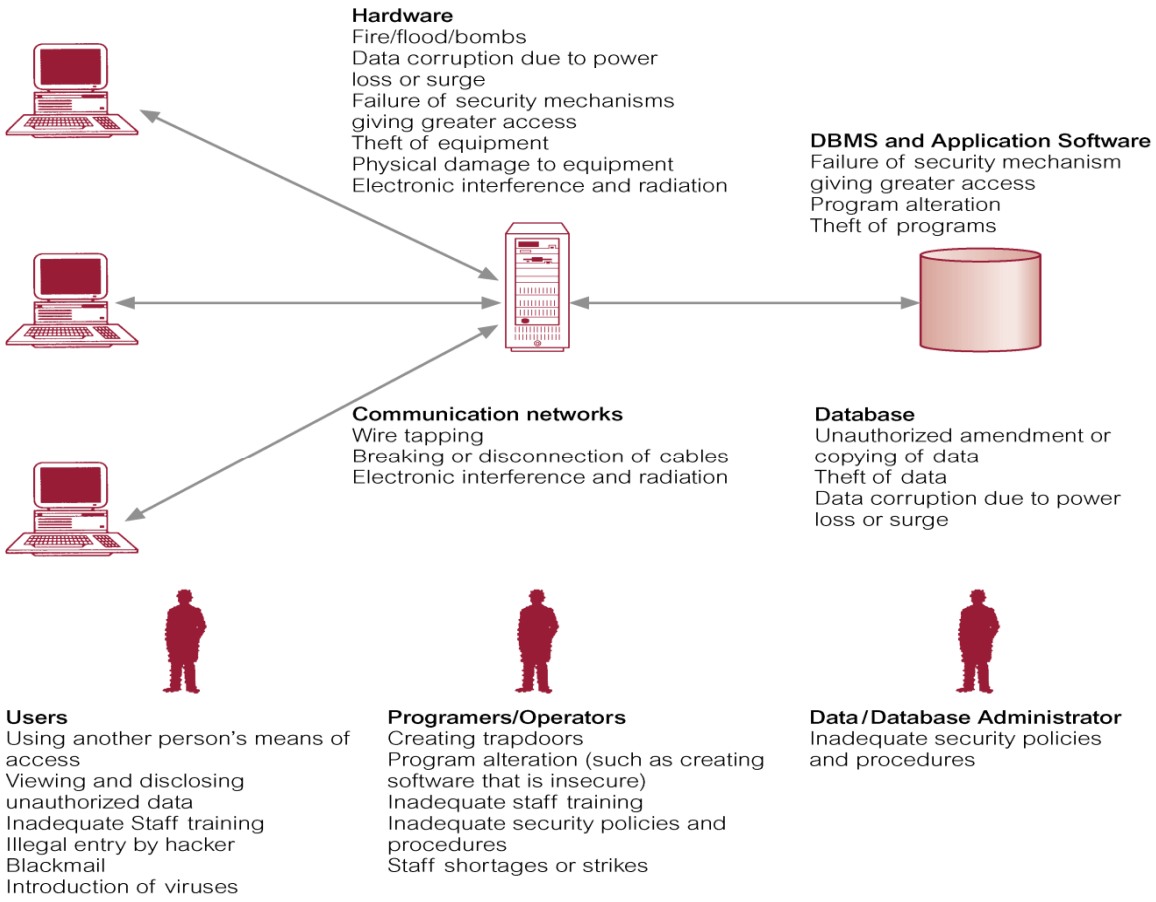

Figure 84 - Summary of Threats to Bophelo Hospital Patient Data. Source: (Connolly & Begg, 2015, p. 611).

Hardware

Any hardware within a computer system is susceptible to failure, either through improper maintenance, outdated technology or physical damage from electrical spikes or impact damage (Munson, 2011). Hardware failure can cause data loss and system downtime. Examples of hardware threats include fires, flooding, bomb detonations, corrupted data due to power surges or power loss, theft of equipment, failure of security mechanisms within the system, electronic interference and radiation, and physical damage done to equipment (Connolly & Begg, 2015).

Users

User threats are caused by any user of the system - whether they are unauthorised or authorised - who can intentionally or unintentionally cause damage to the system or cause it to fail (Connolly & Begg, 2015). User threats are often caused by inadequate staff cyber safety training, whereby a user has not been sufficiently trained to use the computer system and thus cannot avoid potentially dangerous actions; user threats can cause data loss, the spread of malware, hardware failure and security breaches (Connolly & Begg, 2015). Examples of threats caused by users

include gaining access to and disclosing information they are not authorised to, downloading and spreading viruses, modifying system configurations, attackers gaining illegal entry to the system and user susceptibility to social engineering attacks (Connolly & Begg, 2015).

Programmers/Operators

Programmer threats are caused the programmers who design and create the system; these can be caused by programmers not adequately testing their developed systems thus leaving bugs and creating software that does not have the adequate security measures in place (Connolly & Begg, 2015). This can cause data loss, inconsistent databases, system crashes and security breaches. Examples of threats caused by programmers and operators include creating trapdoors in the systems, altering programs, organisational inadequate security policies and procedures, as well as staff shortages and a lack of staff training (Connolly & Begg, 2015).

Database Administrators

Threats caused by database administrators can be caused by the database being designed with inadequate security controls and policies, this can cause data loss, theft or corruption (Connolly & Begg, 2015). Database administrators can fail to keep up-to-date backups of the database which can lead to inconsistencies within the database, or a total loss of the database in the event of a disaster (Connolly & Begg, 2015). Examples of threats caused by database administrators are having inadequate security policies and procedures (Connolly & Begg, 2015).

Communication Networks

A communication network threat is any threat associated with a system being connected to a network, this can cause a number of issues with communication over a system, as well as system downtime if a database is hosted on a shared server on the network; if anything happens to the main server none of the users will be able to access the database and any transactions in process may have been cancelled or corrupted (Connolly & Begg, 2015). Examples of threats caused by communication networks are wiretapping, electronic interference, radiation, and cables being broken or disconnected from the system (Connolly & Begg, 2015).

Database

Database threats are threats to the integrity and consistency of the database, these can cause data loss, corruption and theft (Connolly & Begg, 2015). Examples of database threats are unauthorised changes to data, unauthorised copying of data, data theft and data corruption (Connolly & Begg, 2015).

DBMS and Application Software

Threats caused by a Database Management System (DBMS) and application software refer to software malfunctions, crashes and critical failure resulting in data loss, corruption or unauthorised changes made to the software (Connolly & Begg, 2015). Examples of DBMS and application software threats are failures of security controls and mechanisms allowing for unauthorised access to the system, changes being made to programs, and software theft (Connolly & Begg, 2015).

Database security aims to minimise the losses caused by predicted security breaches in a cost-effective manner without impacting the users of the system (Connolly & Begg, 2015). Bophelo Hospital should be well aware of all threats their system faces, particularly ensuring they are aware of data theft threats as they store confidential patient information in their databases.

QUESTION 2.2.

Countermeasures to threats to the Bophelo Hospital patient data vary from administrative procedures to physical controls on the computer system (Connolly & Begg, 2015). The security of a DBMS is heavily dependent on the security of the operating system - as they are closely associated (Connolly & Begg, 2015).

Below are some of the possible countermeasures to the threats faced by Bophelo Hospital:

Authorisation and Authentication

Authorisation is defined as granting rights or privileges to users of a system to enable them to have legitimate access to either the system or an object on the system (Connolly & Begg, 2015). Authentication is a mechanism that determines where a user is who they claim to be (Connolly & Begg, 2015). Authentication falls under the authorisation process, as any user wishing to access the system must first be identified and authenticated to ensure they are who they claim they are and if they have the correct permissions and access levels to the system or system data (Connolly & Begg, 2015). A system administrator is commonly responsible for authorisation of users by creating accounts for them and granting them specific privileges and roles which identify them and determine what data they are able to see and perform actions on (Connolly & Begg, 2015). This ensures that the database can only be accessed by authorised legitimate users, and acts as a measure of security preventing any data theft or loss (Connolly & Begg, 2015). The authorisation process is as follows: a user logs into the system with their personal credentials, the system authenticates the user's credentials and gets their roles and privileges, the system then authenticates the user's role in the system before allowing them access (Oracle, 2018).

The following diagram outlines the authorisation and authentication process:

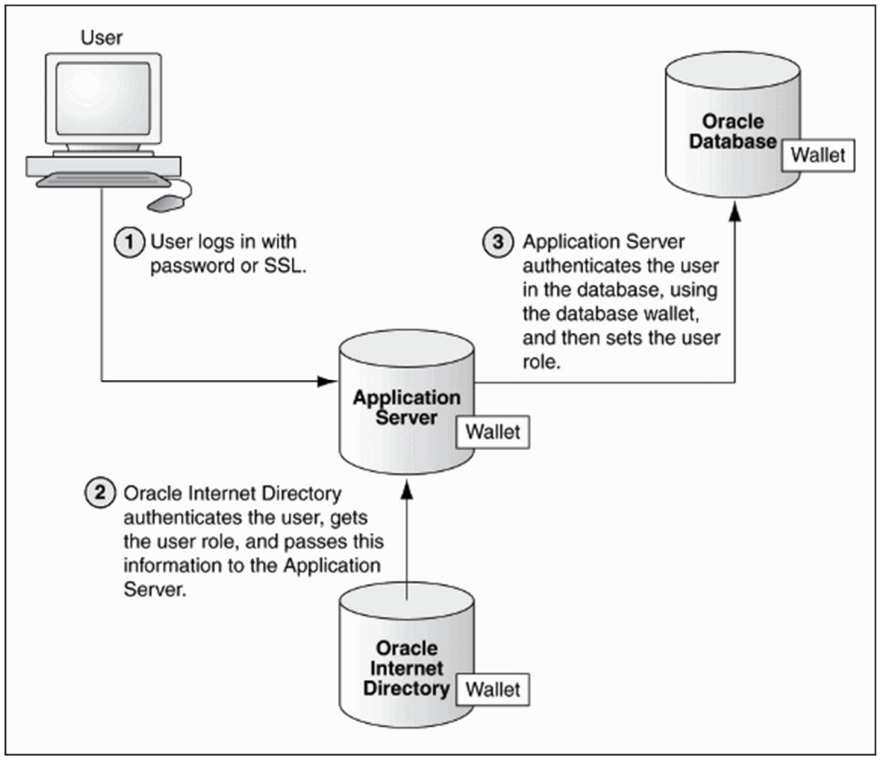

Figure 85 - Authentication and authorisation process on a database system. Source: (Oracle, 2018).

Views

A view is a virtual relation that is produced on request by a user at the time of the request, a view does not actually exist in a database but is created from one or more queries performed on various tables in the database to construct a set of data (Connolly & Begg, 2015). A view is a good security measure for a database system as it allows users to see the data that they require for their work without being granted privileges to use the base relation, the data in a view cannot be altered thus is more secure (Connolly & Begg, 2015).

Backups and Recovery

A backup is a process of copying a database and its log file to offline storage mediums periodically in case of a system failure, it is done to prevent data loss and corruption and to ensure that the database remains consistent and available at all times (Connolly & Begg, 2015). Backups should be made at regular intervals and to a secure storage medium as the backup will be used for recovery in the event of a system failure, if the database becomes unusable or inconsistent the backup will be used to restore it to the most recent stable version with as little data loss as possible (Connolly & Begg, 2015).

Journaling

Journaling is related to backups and recovery, it is the process of keeping a log file that stores all transactions and changes made to the database, in the event of a system failure the log file will be used to assist in the database recovery (Connolly & Begg, 2015). Journaling is useful as it can help system administrator's recovery any lost data changes and transactions that happened between the last backup and a system failure, thus allowing the transactions to be repeated and minimising the data loss and impact of a system failure (Connolly & Begg, 2015).

Integrity

Integrity constraints are used to ensure that all data in the database is valid, this is done to ensure the system is secure and consistent (Connolly & Begg, 2015). Integrity constraints are implemented when data is being entered into the database, all values should be checked that they are correct, and conform to the data type and size of the column they are being inserted into - this is done to ensure that no incorrect results will be outputted by the system (Connolly & Begg, 2015).

Encryption

Encryption involves hiding sensitive data by encoding it using an algorithm to make the data unreadable without the decryption key (Connolly & Begg, 2015). Encryption is a valuable security tool, and can help prevent data theft, as unauthorised users will not be able to read the data without the decryption key; it is suggested to use encryption when transmitting data over a network because of this (Connolly & Begg, 2015). Many DBMS's provide the functionality to encrypt and decrypt data, however, this does slow down the processes because of the time is taken to decrypt the data (Connolly & Begg, 2015).

The following diagram illustrates the encryption process, with encrypted data being stored in the database:

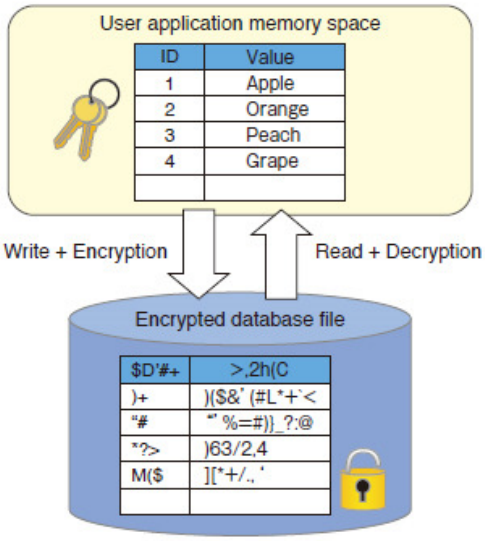

Figure 86 - Database encryption. Source: (Toshiba, 2018).

Redundant Array of Independent Disks (RAID)

RAID is a storage method that ensures a database is fault-tolerant if any hardware component fails the database should still be able to run, thus RAID is used as it provides redundant components that can be integrated into the system in case of a hardware failure (Connolly & Begg, 2015). RAID stores data in such a way that if there is a component failure, no data will be lost as it can be recovered from one of the redundant disks that the data is stored over (Connolly & Begg, 2015). The following diagram illustrates how data is stored in a RAID configuration, with this particular level (RAID 10) storing data in a combined striping and mirroring configuration, the data is nonredundant and mirrored (Connolly & Begg, 2015).

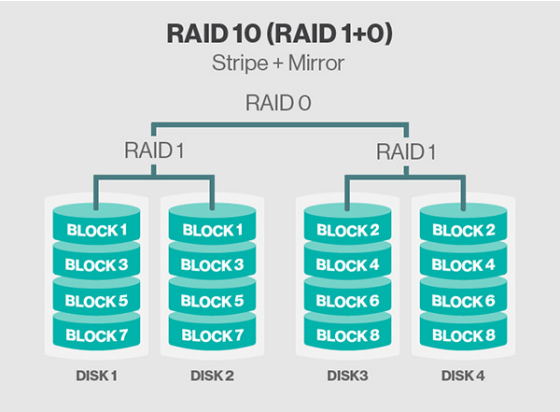

Figure 87 - RAID 10 data storage configurations. Source: (Rouse, 2014).

RAID is a valuable countermeasure for hardware or component failure and prevents any data loss in the event that there is a failure (Connolly & Begg, 2015).

Access Controls

Access controls within a database system are typically provided through the use of user privileges; a privilege allows a user to access - read, write, modify - or create a database object on the DBMS, or to run certain DBMS utilities (Connolly & Begg, 2015). A user should only be granted a privilege if they require it to accomplish a specific task and cannot do so without the role, granting unnecessary privileges can lead to security compromises (Connolly & Begg, 2015). Access controls can be split into two types, mandatory access controls (MAC) and discretionary access controls (DAC). DAC is an approach to managing privileges where the owner of a resource decides how it can be shared, choosing to give other users read or write privileges; whereas MAC is an approach whereby an organisation sets system wide policies to decide how data should be shared, and users cannot change these policies (Connolly & Begg, 2015).

The following diagram outlines the different processes of MAC and DAC:

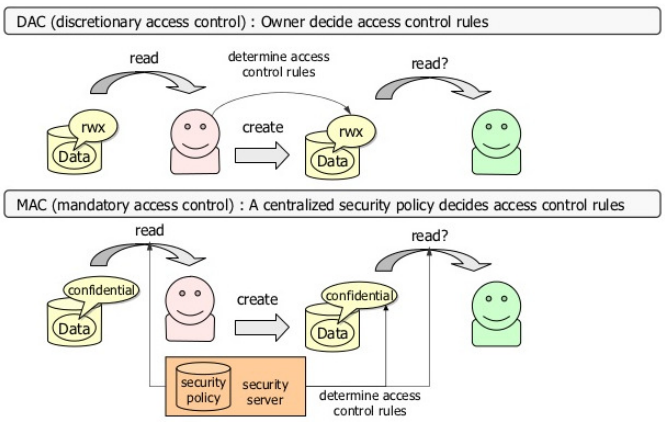

Figure 88 - MAC and DAC processes. Source: (KaiGai, 2012).

The following diagram outlines a multiuser system with security countermeasures in place:

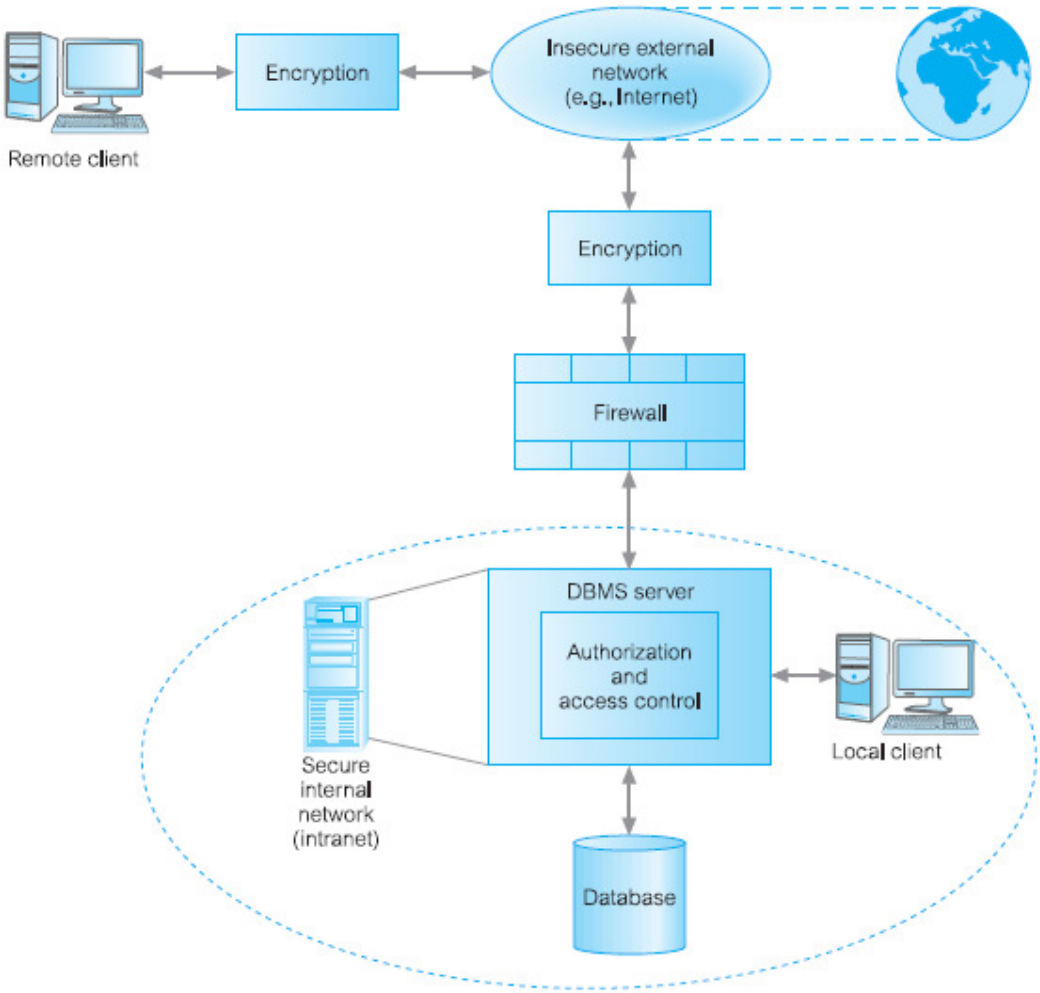

Figure 89 - Security Measures Implemented in a Multiuser System. Source: (Connolly & Begg, 2015, p. 612).

There are various countermeasures Bophelo Hospital can implement to ensure their system is safe from threats, the developers and database administrators should ensure that they are aware of these countermeasures and have implemented them in the Bophelo Hospital System, the diagram shown in Figure 89 is a good example of how to implement security measures in a multiuser system thus should be followed as an example.

QUESTION 2.3.

"Concurrency control is important to maintaining the integrity of a database"

Concurrency control is the process of managing simultaneous operations on a database without the operations interfering with each other (Connolly & Begg, 2015). In a large database system, it is important to allow for multiple users to be able to access shared data simultaneously; if all users have read-only access to the data this system is easy to implement with few issues, however when users need to simultaneously edit and update the shared data issues arise in the system, leading to inconsistency in the database (Connolly & Begg, 2015). Concurrent transactions may threaten the integrity and consistency of the database by having interleaving transactions producing incorrect results, thus the need for concurrency control to be implemented to maintain the integrity of a database (Connolly & Begg, 2015).

The following diagram illustrates where concurrency control fits into a DBMS:

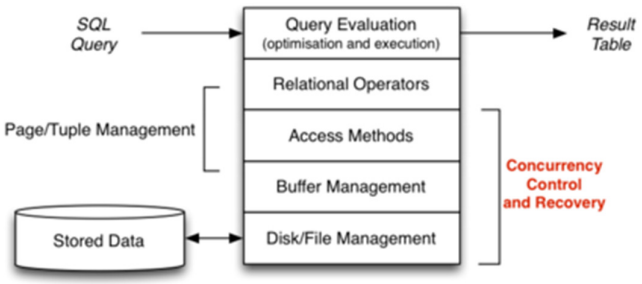

Figure 90 - Concurrency control in a DBMS. Source: (Shepherd, 2016).

Concurrency can cause various potential problems in the database system, these include the lost update problem, the uncommitted dependency problem, and the inconsistent analysis problem (Connolly & Begg, 2015). The lost update problem occurs when a transaction that appears to have been successful is overwritten by another transaction, thus one of the transactions will be lost (Connolly & Begg, 2015). The uncommitted dependency problem, also known as a dirty read, occurs when a transaction reads an uncommitted transaction's intermediate results - thus producing incorrect results for the transaction (Connolly & Begg, 2015). The inconsistent analysis problem occurs when a transaction is in the process of reading values from the database when another transaction update the values as they are being read, thus cause inaccurate and inconsistent data results for the first transaction (Connolly & Begg, 2015).

The following diagram shows the various methods for concurrency control:

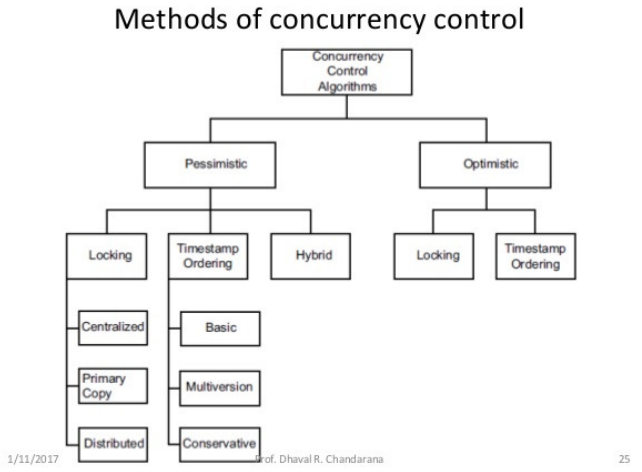

Figure 91 - Concurrency control methods. Source: (Chandarana, 2017).

These problems are caused by concurrency mismanagement leaving the database in an inconsistent state, which is why concurrency controls should be implemented into a database system (Connolly & Begg, 2015). There are two main concurrency control techniques outlined by (Connolly & Begg, 2015), namely locking and timestamping; these are referred to as pessimistic approaches to concurrency control as they delay transactions to ensure that there are no conflicts with other transactions before committing them (Connolly & Begg, 2015).

Locking is a procedure used in concurrency control as a control mechanism for data access, if a transaction is accessing a database, that object being accessed will be locked to deny access to other transactions trying to access it - this is to prevent incorrect results and inconsistency within the database (Connolly & Begg, 2015). The disadvantage of locking is that it can lead to a deadlock problem - where two or more transactions are waiting for one another for their locks to be released when they're held by the other (Connolly & Begg, 2015). Deadlocks can be prevented by the other concurrency control method being discussed - timestamping.

The following diagrams illustrate a pessimistic locking process:

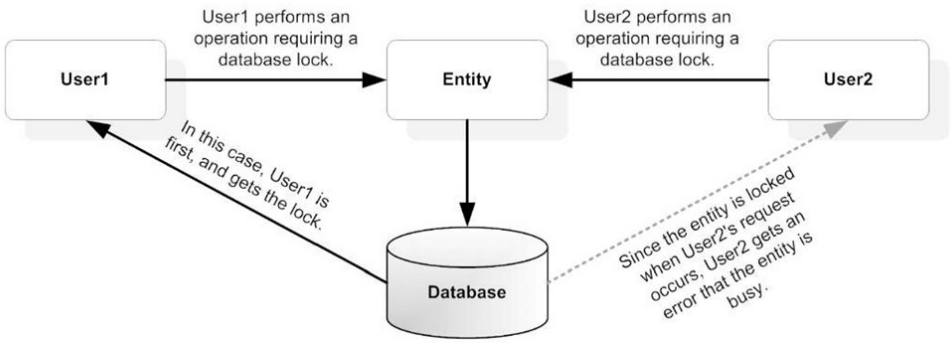

Figure 92 - Pessimistic locking approach to concurrency control

Timestamping is a concurrency control approach where transactions are ordered in a way where older transactions with a smaller timestamp get priority in the event of a transaction conflict; timestamps are a unique identifier given to a transaction by a DBMS that indicate when the transaction began (Connolly & Begg, 2015). If a transactions attempts a read or write action on the database, it will only be allowed to read or write if the last update to that data item was done by another transaction with an older timestamp to prevent concurrent transaction conflicts, if the data item is currently in use by a transaction, the transaction attempting to read/write to the item will be restarted and given a new timestamp (Connolly & Begg, 2015). Timestamping uses a protocol called basic timestamp ordering to ensure that transactions are conflicted serializable, and the transactions are carried out in a chronological order of the timestamps (Connolly & Begg, 2015).

The following diagram illustrates the timestamping process, showing the last committed timestamp format and the committed transactions new timestamp:

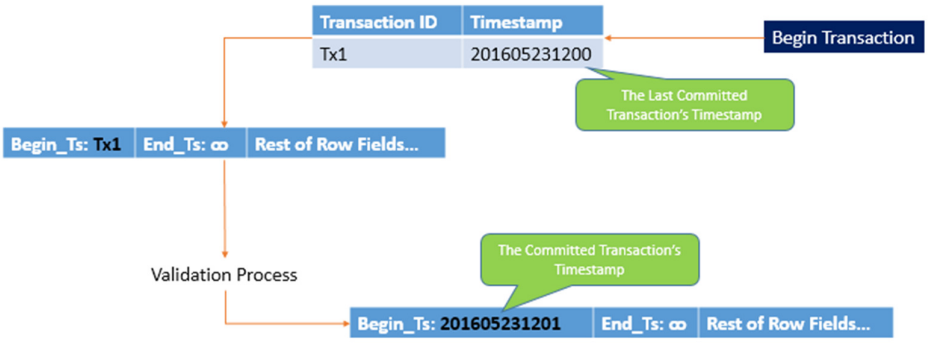

Figure 93 - Timestamping approach to concurrency control. Source: (Fard, 2014).

The advantages of concurrency control for database transactions are:

- Various concurrency problems are avoided, such as dirty reads, lost updates, inconsistent analysis and phantom reads (Connolly & Begg, 2015).
- Ensures database integrity and consistency (Connolly & Begg, 2015).

The disadvantages of concurrency control for database transactions are:

- Transactions may take longer to be processed depending on the concurrency control method being used (Connolly & Begg, 2015).
- Issues may arise from the concurrency control methods, such as a deadlock problem being created when using the locking approach to concurrency control (Connolly & Begg, 2015).

Concurrency control is needed in any system where multiple users access the database simultaneously to ensure that data integrity is maintained; concurrency controls prevent any problems that arise from simultaneous transactions in the system - such as the lost update problem, uncommitted dependency problem, and inconsistent analysis problems (Connolly & Begg, 2015). The concurrency control methods that should be used to maintain the integrity of the database are locking and timestamping. A database should be consistent and reliable, concurrency control ensures that the database is consistent by ensuring the integrity of the data on a database and managing the transactions that affect the database (Connolly & Begg, 2015).

QUESTION 2.4.

Database recover is the process of returning a database to its original stable state in the event of a failure wherein the database has lost data or is no longer functional (Connolly & Begg, 2015). Database recovery should be built into the DBMS that is being used to ensure that the database is reliable, and in the correct state at all times - this means that the database should be resilient to failures and should be able to recover from any failures that do occur (Connolly & Begg, 2015).

Data is stored on various types of media, each having different levels of reliability - the main storage mediums being referred to as primary storage, or main memory, which is volatile and does not recover well from crashes, and secondary storage, which is more reliable but significantly slower (Connolly & Begg, 2015). There are various types of failure which each affects different storage mediums, thus needing to be dealt with in different ways depending on how it affects the storage of data (Connolly & Begg, 2015).

No matter what the cause of failure is, database recovery focuses on reducing the effects of the loss of main memory - which holds the database buffers - and the loss of the disk copy of the database (Connolly & Begg, 2015).

The various causes of database failure include the following:

System Crashes

System crashes can be caused by both hardware and software failures or errors; these can result in a loss of main memory (Connolly & Begg, 2015).

Media Failures

Media crashes commonly include unreadable media and head crashes; these can result in partial loss of data stored in secondary storage (Connolly & Begg, 2015).

Application Software Errors

Application software errors, such as logical errors in a program that uses and accesses the database can cause one or multiple failed transactions in the database (Connolly & Begg, 2015).

Natural Physical Disasters

Natural physical disasters can include fires, earthquakes, floods, or power failures; these can cause equipment failure or loss which in turn can lead to data loss (Connolly & Begg, 2015).

Carelessness

Carelessness can cause the unintentional destruction of data or hardware by users or operators, this can lead to loss of equipment and data (Connolly & Begg, 2015).

Sabotage

Sabotage or the intentional corruption and destruction of data, hardware, or software can lead to equipment failure, loss and data loss (Connolly & Begg, 2015).

The following diagrams show the causes and the average cost of data loss:

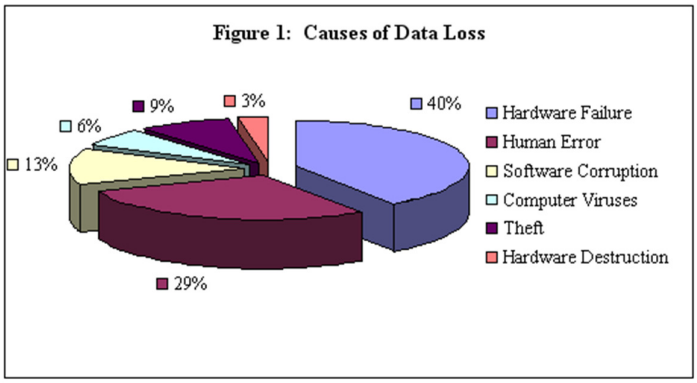

Figure 94 - Causes of Data Loss. Source: (Smith, 2003).

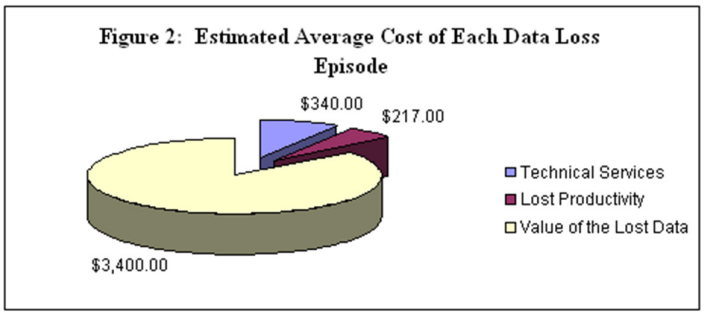

Figure 95 - Average Cost of Data Loss Episodes in USD. Source: (Smith, 2003).

As shown in the above figures, data loss is a high cost incident, (Smith, 2003) estimates that data loss cost the US $18.2 Billion (R226.3 Billion) annually - see Figure 96 -'s calculations for the annual US cost of data loss., thus database recovery is a vitally important process within an organisation.

Table 2: Total US Data Loss Estimates

Types of Loss	Average Cost of Each Data Loss Incident
Technical Services	340
Lost Productivity	217
Value of the lost data	3,400
Sub Total	$3,957
Total US Data Loss Costs	$18.2 Billion

Source: Author's estimates based on data referenced in all prior tables and figures.

Figure 96 - (Smith, 2003)'s calculations for the annual US cost of data loss.

Database recovery takes place if a database failure has occurred and there were data losses within the database, or if the entire database was lost or corrupted - the type of recovery technique depends on the type of failure that occurred (Connolly & Begg, 2015). A DBMS should provide backup mechanisms, logging facilities, checkpointing facilities, and recovery managers to assist with recovery in the event of a failure, these are explained below (Connolly & Begg, 2015).

The following facilities should be provided by a DBMS to support recovery:

Backup Mechanism

Backup copies of the database should be made at regular intervals without interrupting the system progress or functionality; the backup copy of the database can be used to restore the database in the event of a system failure or if any damage is done to the database (Connolly & Begg, 2015). Backups can be a full copy of the database, or incremental copies where only the latest changes since the last full backup are saved - backups are typically stored on offline storage such as hard disks or optical disks for easier access and to ensure there is minimal downtime (Connolly & Begg, 2015).

Logging Facilities

A DBMS should keep track of database transactions through the use of a log (or journal) file that stores information about all updates to the database (Connolly & Begg, 2015). The log can contain transaction records and checkpoint records; transaction records contain the transaction identifier, type of log record, identifier of the data item affected by the database action, a before and after image of the data item and its value before and after being changed, and any log management information such as pointers to the previous and next logs for the transactions (Connolly & Begg, 2015).

Checkpoint Facility

Checkpointing is the point of synchronisation between the database and the transaction log file (Connolly & Begg, 2015). Checkpointing is performed alongside logging to ensure that if there are any errors or failures occur, the database administrator will be able to find how far back in the log to search and avoid redoing safely written transactions; checkpointing limits the amount of searching and processing that is needed to carry out the log file (Connolly & Begg, 2015). Checkpoints take place at scheduled intervals and perform the following functions: copying all log records from main memory to secondary storage; writing the modified blocks in database buffers to the secondary storage; and writing a checkpoint record to the log file (Connolly & Begg, 2015).

The recovery technique used is dependent on the damage was done to the database, (Connolly & Begg, 2015) outlines two cases:

- The database has been extensively damaged or destroyed;
- The database has not been physically damaged but is inconsistent.

If the database has been extensively damaged or destroyed, it should be restored from a backup copy of the database and updates should be made to it using the saved log file, assuming the log file was not damaged along with the database - for this reason databases and log files should always be stored in separate files (Connolly & Begg, 2015).

In the case of a database being in an inconsistent state the following techniques can be used to recovery any lost data when restoring a backup of a database:

Deferred Update

The deferred update technique means that updates will not be written to the disk until after a transaction has reached its commit point if the transaction fails to reach the commit point the database will not be modified (Connolly & Begg, 2015). When a transaction begins, a *transaction start* is written on the log, and all subsequent updates are recorded in the log, before the commit point is reached the updates are recorded in the local database buffers, and after the commit point the updates are applied to the database and a *transaction commit* is written to the log record (Connolly & Begg, 2015) (Shukla, 2010). If the transaction fails before the commit point, the log record will be ignored, and nothing will be written to the database, so if a system failure occurs during the transaction the database administrator can review the log records to reapply the updates to the system and no further damage or corruption is done to the data stored in the database (Connolly & Begg, 2015).

The following diagram illustrates the deferred update process:

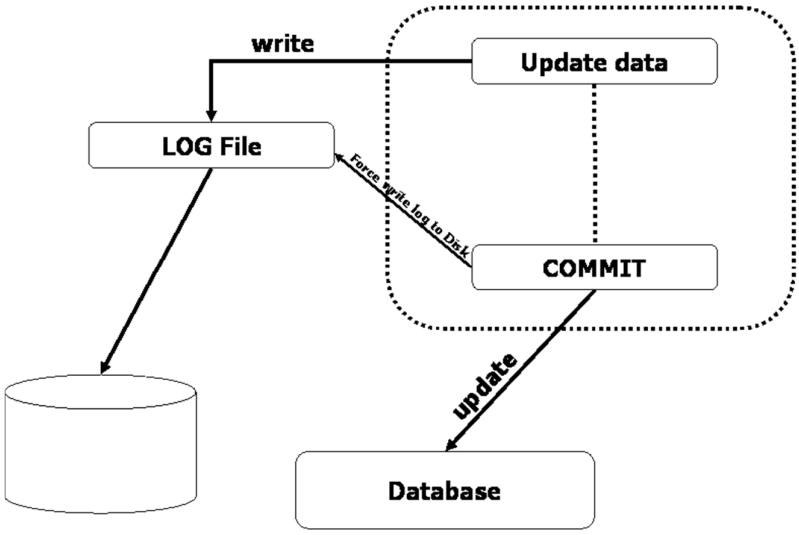

Figure 97 - Deferred update process. Source: (Shukla, 2010).

Immediate Update

The immediate update protocol process involves updates being made to the database immediately as they occur, without waiting to reach the commit point, all transactions that are being performed are written to a log file for recovery (Connolly & Begg, 2015) (Shukla, 2010). When a transaction begins, a *transaction start* should be written to the log, as a write operation occurs a record should be saved to the log file, when the log record has been written the database buffers can be updated, once the buffers are flushed to secondary storage the database is then updated with the write operation, and when the transaction is committed a *transaction commit* is written to the log (Connolly & Begg, 2015). If a failure occurs during the transaction the log file will be needed to undo all previous versions with incorrect values, and the transaction will need to be redone using the log file (Connolly & Begg, 2015) (Shukla, 2010).

The following diagram illustrates the immediate update process:

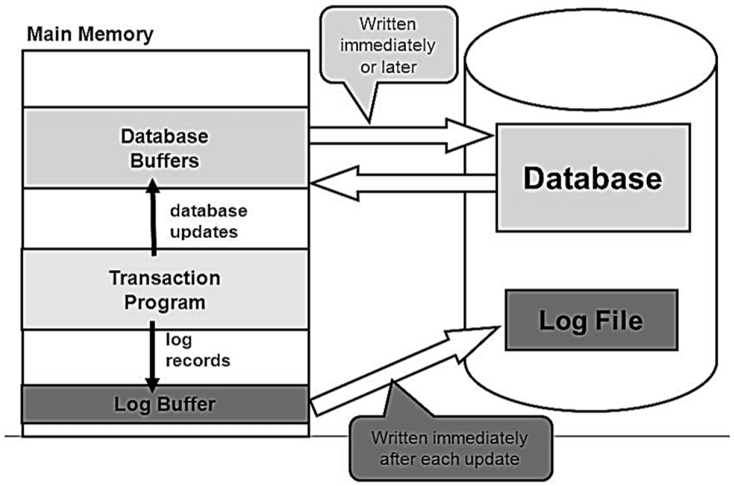

Figure 98 - Immediate update process. Source: (Watts, 2016).

Log files are used for both immediate and deferred update recovery of an inconsistent database.

Shadow paging

An alternative to log-based recovery techniques, as suggested by (Lorie, 1977) is a technique called shadow paging; whereby during a transaction life cycle two tables are held - a current page table and a shadow page table (Connolly & Begg, 2015). When the transaction begins, both tables are the same, and the shadow page table will not change at any point throughout the transaction thus will be used to restore the database in the event of a failure (Connolly & Begg, 2015). The current table will record all updates to the database, and when the transaction is complete the current table will replace the shadow table (Connolly & Begg, 2015). Shadow paging offers the advantages of having faster recovery time with no need for undo and redo operations and having less overhead due to having no log files to maintain; however, the disadvantages are that data fragmentation can occur, and blocks can become inaccessible requiring garbage collection to reclaim access to them (Connolly & Begg, 2015).

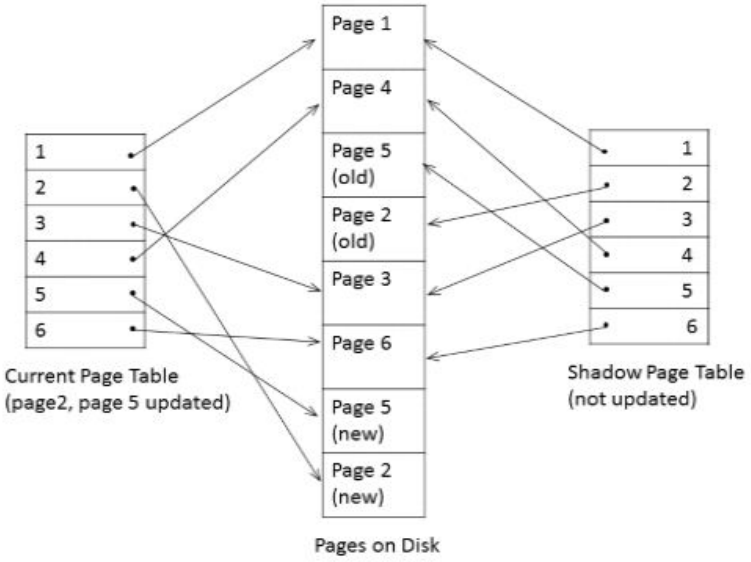

Figure 99 - Shadow paging process. Source: (Francis, 2016).

Database recovery is vital to any system as failures can be unpreventable and unpredictable - databases must always remain consistent and reliable to be valuable to an organisation (Connolly & Begg, 2015). Tsakani Retail Outlets should ensure that they are aware of the causes of failure in order to choose the right recovery technique for their database system.

REFERENCES

1 Key Data, 2018. *Conceptual Data Model.* [Online]
Available at: https://www.1keydata.com/datawarehousing/conceptual-data-model.html
[Accessed May 2018].

1 Key Data, 2018. *Logical Data Model.* [Online]
Available at: https://www.1keydata.com/datawarehousing/logical-data-model.html
[Accessed May 2018].

1 Key Data, 2018. *Physical Data Model.* [Online]
Available at: https://www.1keydata.com/datawarehousing/physical-data-model.html
[Accessed May 2018].

Alexandr, M., 2018. *MySQL Trigger Validation Example.* [Online]
Available at: https://mac-blog.org.ua/mysql-trigger-validation-example/
[Accessed May 2018].

Ali, K., 2012. *How to get ER model of database from server with Workbench.* [Online]
Available at: https://stackoverflow.com/questions/9589727/how-to-get-er-model-of-database-from-server-with-workbench?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
[Accessed May 2018].

Burtescu, E., 2009. Database Security - Attacks and Control Methods. *Journal of Applied Quantitive Methods,* 4(4), pp. 449-454.

Chandarana, D., 2017. *Distributed DBMS - Unit 8 - Distributed Transaction Management & Concurrency Control.* [Online]
Available at: https://www.slideshare.net/DhavalChandarana/distributed-dbms-unit-8-distributed-transaction-management-concurrency-control
[Accessed May 2018].

Connolly, T. & Begg, C., 2015. *Database Systems: A Practical Approach to Design, Implementation, and Management.* 6th ed. Harlow: Pearson.

Fard, H. J., 2014. *SQL Server In-Memory Multi-Version Concurrency Control.* [Online]
Available at: http://fard-solutions.com/sql-server-memory-multi-version-concurrency-control/
[Accessed May 2018].

Francis, E., 2016. *Data Concurrency Control And Data Recovery.* [Online]
Available at: http://slideplayer.com/slide/7393128/
[Accessed May 2018].

KaiGai, K., 2012. *Label based Mandatory Access Control on PostgreSQL.* [Online]
Available at: https://www.slideshare.net/kaigai/label-based-mandatory-access-control-on-postgresql
[Accessed May 2018].

Lafferty, P., 2017. *MySQL Stored Procedures 101.* [Online]
Available at: https://medium.com/@peter.lafferty/mysql-stored-procedures-101-6b4fe230967
[Accessed May 2018].

Lorie, R. A., 1977. Physical Integrity in a Large Segmented Database. *ACM Transactions on Database Systems,* 2(1), pp. 91-104.

Munson, L., 2011. *WHAT ARE THE TOP REASONS FOR HARDWARE FAILURE AND DATA LOSS?.* [Online]
Available at: http://www.security-faqs.com/the-top-reasons-for-hardware-failure-and-data-loss.html
[Accessed May 2018].

MySQL Tutorial, 2018. *Getting Started with MySQL Stored Procedures.* [Online]
Available at: http://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx
[Accessed May 2018].

MySQL, 2018. *Adding an EER Diagram.* [Online]
Available at: https://dev.mysql.com/doc/workbench/en/wb-creating-eer-diagram.html
[Accessed May 2018].

MySQL, 2018. *How to Get the Unique ID for the Last Inserted Row.* [Online]
Available at: https://dev.mysql.com/doc/refman/8.0/en/getting-unique-id.html
[Accessed May 2018].

MySQL, 2018. *Privileges Provided by MySQL.* [Online]
Available at: https://dev.mysql.com/doc/mysql-security-excerpt/5.7/en/privileges-provided.html
[Accessed May 2018].

MySQL, 2018. *The Schema Privileges Pane.* [Online]
Available at: https://dev.mysql.com/doc/workbench/en/wb-model-overview-privileges.html
[Accessed May 2018].

Oracle, 2002. *Privileges, Roles, and Security Policies.* [Online]
Available at: https://docs.oracle.com/cd/B10501_01/server.920/a96524/c24privs.htm
[Accessed May 2018].

Oracle, 2018. *Configuring Authentication.* [Online]
Available at: https://docs.oracle.com/cd/E11882_01/network.112/e36292/authentication.htm#DBSEG30031
[Accessed May 2018].

Rayan, J. C., 2015. *Data modelling using ERD with Crow Foot Notation.* [Online]
Available at: https://www.codeproject.com/Articles/878359/Data-modelling-using-ERD-with-Crow-Foot-Notation
[Accessed May 2018].

Rouse, M., 2014. *RAID 10 (RAID 1+0).* [Online]
Available at: https://searchstorage.techtarget.com/definition/RAID-10-redundant-array-of-independent-disks
[Accessed May 2018].

Seth, L., 2015. *Connecting Java Application to MySQL Database in NetBeans.* [Online]
Available at: http://www.lionblogger.com/java-application-mysql-database/
[Accessed May 2018].

Shepherd, J., 2016. *Transactions, Concurrency, Recovery.* [Online]
Available at: https://webcms3.cse.unsw.edu.au/COMP9315/16s1/resources/2352
[Accessed May 2018].

Shukla, P., 2010. *Defferred Update method.* [Online]
Available at: https://dbms-ii.blogspot.co.za/2010/03/defferred-update-method.html
[Accessed May 2018].

Shukla, P., 2010. *Immediate Update Method.* [Online]
Available at: https://dbms-ii.blogspot.co.za/2010/03/immediate-update-method.html
[Accessed May 2018].

Singh, S. K., 2011. *Database Systems: Concepts, Design and Applications.* 2nd ed. Delhi: Pearson Education India.

Smith, D. M., 2003. The Cost of Lost Data. *Graziadio Business Review,* 6(3).

Toshiba, 2018. *https://www.toshiba.co.jp/rdc/rd/fields/11_e05_e.htm.* [Online]
Available at: https://www.toshiba.co.jp/rdc/rd/fields/11_e05_e.htm
[Accessed May 2018].

Vuorinen, C., 2013. *Validating data with triggers in MySQL.* [Online]
Available at: https://cvuorinen.net/2013/05/validating-data-with-triggers-in-mysql/
[Accessed May 2018].

W3 Schools, 2018. *SQL CREATE INDEX Statement.* [Online]
Available at: https://www.w3schools.com/sql/sql_create_index.asp
[Accessed May 2018].

W3Resource, 2018. *MySQL Stored Procedure.* [Online]
Available at: https://www.w3resource.com/mysql/mysql-procedure.php
[Accessed May 2018].

W3Schools, 2018. *SQL Inner Join.* [Online]
Available at: https://www.w3schools.com/sql/sql_join_inner.asp
[Accessed May 2018].

Watts, E., 2016. *Chapter 17: Recovery..* [Online]
Available at: http://slideplayer.com/slide/7726203/
[Accessed May 2018].

Wenzel, K., 2018. *What is a Database Trigger?.* [Online]
Available at: https://www.essentialsql.com/what-is-a-database-trigger/
[Accessed May 2018].

Zaiane, O., 1998. *Integrity Constraints.* [Online]
Available at: http://www.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chapter6/node1.html
[Accessed May 2018].