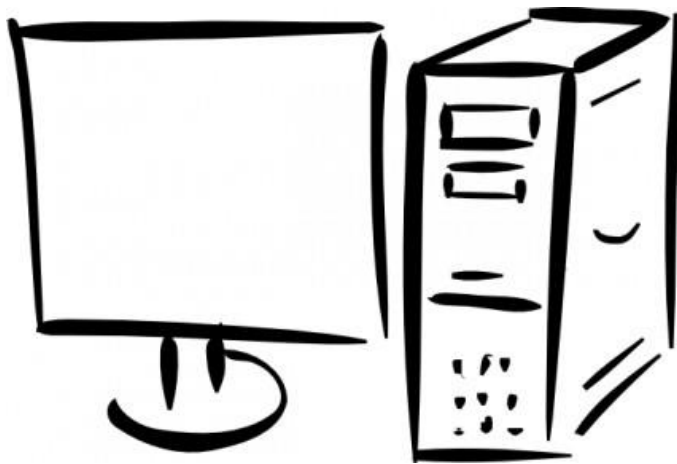


Advanced Database System Project

ITDA310 - 2018



Matthew Van der Bijl
Xq9x3wv31

Table of Contents

Introduction	1
Question 1	2
1.1 Database Design	3
a) Conceptual Design	3
Identify Entity Types	3
Identify Relationships	4
Identify & Associate Attributes	6
Hobby Table Attributes	6
Users Table Attributes	6
Message Table Attributes	7
Match Table Attributes	7
Gender Table Attributes	7
b) Logical & Physical Design.....	9
Hobby Table Design.....	9
Single Person Table Design	10
Person Hobby Table Design.....	12
Message Table Design.....	13
Compatibility Table Design	15
Gender Table Design	16
c) Fully Function Database	18
Create Statements	19
Stored Procedures	22
Match Users Stored Procedure	22
Add Hobby Stored Procedures.....	23
Count Messages Stored Procedure	24
Triggers.....	25
Indices.....	26

Message Table Index.....	27
Gender Table Index	28
Hobby Table Index.....	29
Single Person Table Index	30
Compatible Table Index	30
Views	31
Get Messages View	31
Get hobbies View.....	33
Final ERD.....	35
Prototypes	36
Code Snippets.....	38
Final Interfaces.....	41
1.2 Roles & Privileges	45
a) Roles	45
b) Global & Schema Privileges.....	47
Developer Privileges.....	47
User Privileges	50
Exporting Data.....	53
Conclusion	54
Question 2	55
2.1	55
2.2	58
2.3.....	61
2.4	65
Bibliography	68

Table of Tables

Table 1 Attributes associated with a hobby	6
Table 2 Attributes associated with a user	6
Table 3 Attributes associated with a message	7
Table 4 Attribute associated with a match	7
Table 5 Attributes associated with a gender	7
Table 6 Properties of hobby table.....	9
Table 7 Properties of the single person table	10
Table 8 Properties of person hobby linkage table.....	12
Table 9 Properties of message table	13
Table 10 Properties of compatibility table.....	15
Table 11 Properties of gender table	16
Table 12 Users & Roles	45
Table 13 Table of Roles	46
Table 14 Common causes of database failure	65



Table o Figures

Figure 1 Entities of the new system.....	4
Figure 2 - ERD containing entities & relationships.....	5
Figure 3 - Full ERD	8
Figure 4 SQL Script to create Hobby Table.....	9
Figure 5 SQL script to create single person table.....	11
Figure 6 SQL script to create person hobby table	12
Figure 7 SQL script to create message table	14
Figure 8 SQL script to create compatible table.....	15
Figure 9 SQL script to create gender table.....	16
Figure 10 ERD with attributes & data types.....	17
Figure 11 Creating the database.....	18
Figure 12 Creating the Compatible table.....	19
Figure 13 Creating the Gender table.....	19
Figure 14 Creating the Hobby table.....	19
Figure 15 Creating the Message table	20
Figure 16 Creating a table to link a person to the hobby	20
Figure 17 Creating the user table.....	21
Figure 18 Match User stored Procedure.....	22
Figure 19 SQL script used to create hobby procedure	23
Figure 20 Add Hobby stored Procedure	23
Figure 21 count messages stored procedure	24
Figure 22 Trigger to validate phone number (only numbers)	25
Figure 23 Trigger to validate message ID.....	25
Figure 24 Indices within the database	26
Figure 25 Message table index	27
Figure 26 Query to create an index.....	27
Figure 27 Index on gender table	28

Figure 28 SQL Script used to create an index on gender table	28
Figure 29 Index on hobby table	29
Figure 30 SQL script to create an index on hobby table	29
Figure 31 Index on single person table	30
Figure 32 Index on compatible table	30
Figure 33 Creating a get_message view	31
Figure 34 Creating a get_messages view	31
Figure 35 SQL Script used to create a get_messages view	32
Figure 36 Creating get a view to get hobbies	33
Figure 37 Creating get hobbies view	33
Figure 38 SQL script used to create get hobbies view.....	34
Figure 39 ERD generated by Workbench.....	35
Figure 40 Prototype Message Panel	36
Figure 41 Prototype Main Frame.....	36
Figure 42 Prototype Information Editor.....	37
Figure 43 Prototype Registration.....	37
Figure 44 Java database connection.....	38
Figure 45 Java code to update the database	38
Figure 46 Java code to insert into the database	39
Figure 47 Java code to delete from the database.....	39
Figure 48 Java code to call stored procedures	40
Figure 49 Edit information panel	41
Figure 50 Select profile picture dialogue	41
Figure 51 View user details panel	42
Figure 52 View image dialogue	42
Figure 53 Matches panel.....	43
Figure 54 Message panel.....	43
Figure 55 Logoff dialogue	44

Figure 56 Exit dialogue	44
Figure 57 Users and Roles	45
Figure 58 All users on the database.....	47
Figure 59 Details of the developer user.....	48
Figure 60 Details of developer user.....	48
Figure 61 Global privileges of the developer user	49
Figure 62 Privileges of the developer user	49
Figure 63 Login details of a generic user.....	50
Figure 64 Account limits for a generic user	50
Figure 65 Roles for a generic user	51
Figure 66 Privileges of a generic user	51
Figure 67 Schema privileges of a generic user.....	52
Figure 68 Exporting database	53
Figure 69 Successfully exported database.....	53
Figure 70 Database threats.....	56
Figure 71 Layer of Security (Whitman, et al., 2012)	58
Figure 72 Database authorisation and authentication (Kroenke & Auer, 2013)	59
Figure 73 Lost update example (Connolly & Begg, 2015)	61
Figure 74 Inconsistent analysis problem (Connolly & Begg, 2015)	62
Figure 75 Cascading rollback (Connolly & Begg, 2015)	63



Introduction

The project involves the creation of a Java Swing based dating application. The project will be created using NetBeans. It is required that the project makes use of a MySQL database. Development of the database will be done using phpMyAdmin and MySQL Workbench. The dating application itself will need to interact with the database. The database needs to be fully functional, secure and make use of modern database design principles and conventions. The application will need to make use of object-oriented programming principles as well as modern design principles. The final system will need to provide users with a holistic User Experience (UX).

The project's source code, Javadoc, accompanying SQL script and design documentation will be submitted as well as a Turnitin report. The entire project will be made available on GitHub.

The submission that follows outlines the project's development plan, diagrams/models and prototypes interfaces, design considerations and implementation plan. The final aspect of this document is theory questions provided by the examiner.

The source code and all accompanying documentation used will be made available on GitHub.

Question 1

The need for the creation of a new online dating application (app) has been created by the decline in credibility of existing dating apps. This app is designed will be integrated with a fully functional MySQL database and can be created with any programming language in any system. The system will make use of a MySQL database to store data and will be interacted with using Structured Query Language (SQL). This new online dating app will be developed by following design methodologies presented by Connolly and Begg (2015). The system will need to ensure entity integrity.

Connolly and Begg (2015) define a design methodology as the usage of procedures, tool, techniques and documentation in a structured manner to assist in the design and development process. Connolly and Begg (2015) suggested the following steps when developing an information system:

1. Identify Entity Types;
2. Identify Relationships;
3. Identify & Associate Attributes;
4. Identify Attributes domain;
5. Remove Redundancy;
6. Validate Conceptual models; and;
7. Review Conceptual model.

An adaptation of Connolly and Begg (2015) suggested methodology has been used in the development of the online dating system. All user interfaces (UI) and other interactive systems have been designed following design principles and heuristic considerations suggested by Rogers, et al. (2011).

1.1 Database Design

a) Conceptual Design

The objective of creating a conceptual database model is to establish all enterprise data requirements (Connolly & Begg, 2015). The conceptual data model should include Entity Relationship Diagrams (ERDs) and any other supporting documentation (Connolly & Begg, 2015). The section that follows will outline all possible entities, entity relationships and a normalised ERD.

Identify Entity Types

According to Connolly and Begg (2015), the purpose of this step is to identify the entities that the new system will require.

Entities:

1. Hobby;
2. Users;
3. Message;
4. Compatibility; and;
5. Gender.

These entities will make up the database.

The figure below is a graphical representation of the entities in the new system. The structure presented is based on a format presented by Connolly and Begg (2015). A JPG image of this ERD will be submitted alongside this document.

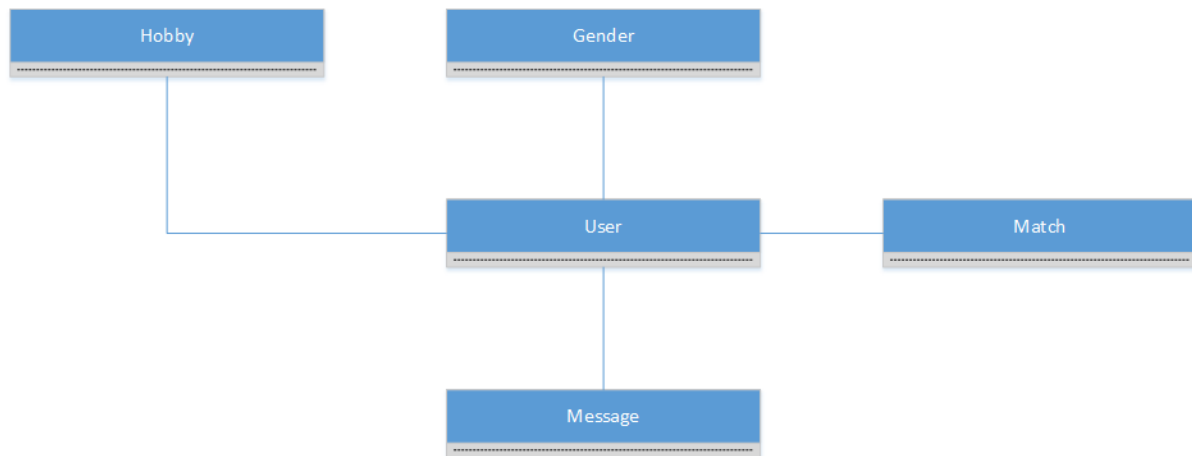


Figure 1 Entities of the new system

Please note that the above ERD only shows the entities of the system and not cardinality, multiplicity or attributes.

Identify Relationships

According to Connolly and Begg (2015), it is important to clearly establish the relationships between the various entities that make up a system. The list below identifies some important relationships that exist between entities:

1. Hobby and users will have a many-to-many relationship (multiple hobbies, multiple users).
2. User and message will have a many-to-many relationship (multiple users, multiple messages).
3. User and matches will have a many-to-many relationship (multiple users, multiple matches).
4. User and gender will have a one-to-one relationship (one user, one gender).

Due to the limitation on the relational database model, its lack of support for many-to-many relationships, an additional entity is required to link the user entity to the hobby entity their relationship (Connolly & Begg, 2015).

The figure below is a graphical representation of the entities and their relationships. The structure presented is based on a format presented by Connolly and Begg (2015). A JPG image of this ERD will be submitted alongside this document.

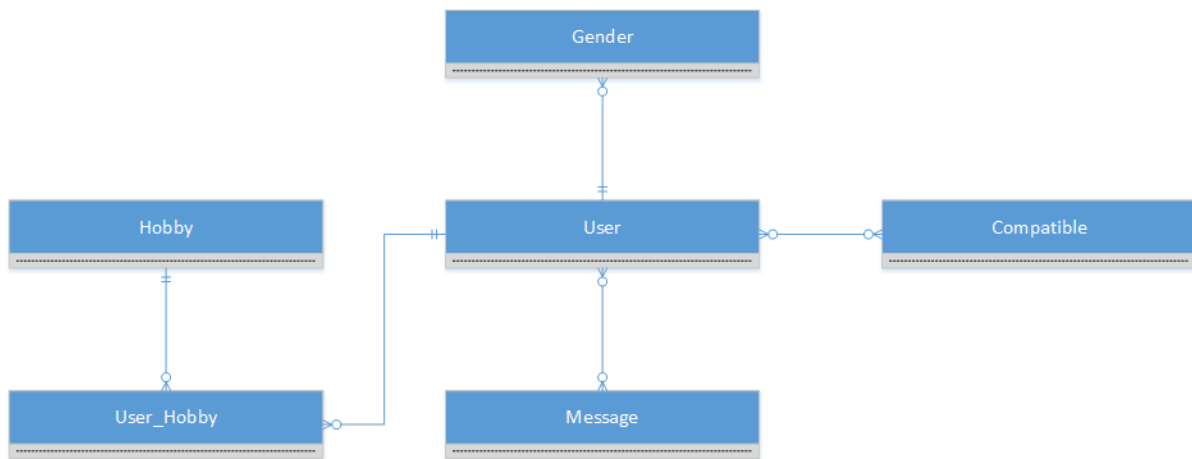


Figure 2 - ERD containing entities & relationships

Please note that the above ERD does not indicate attributes.

Identify & Associate Attributes

The tables below outline each database table's (relation) attributes.

Hobby Table Attributes

Table 1 Attributes associated with a hobby

Attribute Name	Attribute Description
ID	Used to uniquely identify each hobby.
Name	The name of the hobby.
Description	A brief description of what the hobby entails.

Users Table Attributes

Table 2 Attributes associated with a user

Attribute Name	Attribute Deception
ID	Used to uniquely identify a user.
Username	Username used for login.
First name	The user's first name.
Surname	The user's last name.
Phone number	The user's phone number.
Last login timestamp	The data and time that the user last logged in.
Gender	The user's gender.
Profile picture	Picture of the user.

Message Table Attributes

Table 3 Attributes associated with a message

Attribute Name	Attribute Description
ID	User to uniquely identify a message.
Message	The message that was sent.
From	The user that sent the message.
To	The users who need to receive the message.

Match Table Attributes

Table 4 Attribute associated with a match

Attribute Name	Attribute Description
User A	One of the two users who matched.
User B	One of the two users who matched.

Gender Table Attributes

Table 5 Attributes associated with a gender

Attribute Name	Attribute Description
Name	The name of the gender. Used to uniquely identify each gender.
Description	A brief description of the gender.

The ERD that follows is a complete ERD for the entire system. The structure presented is based on a format presented by Connolly and Begg (2015).

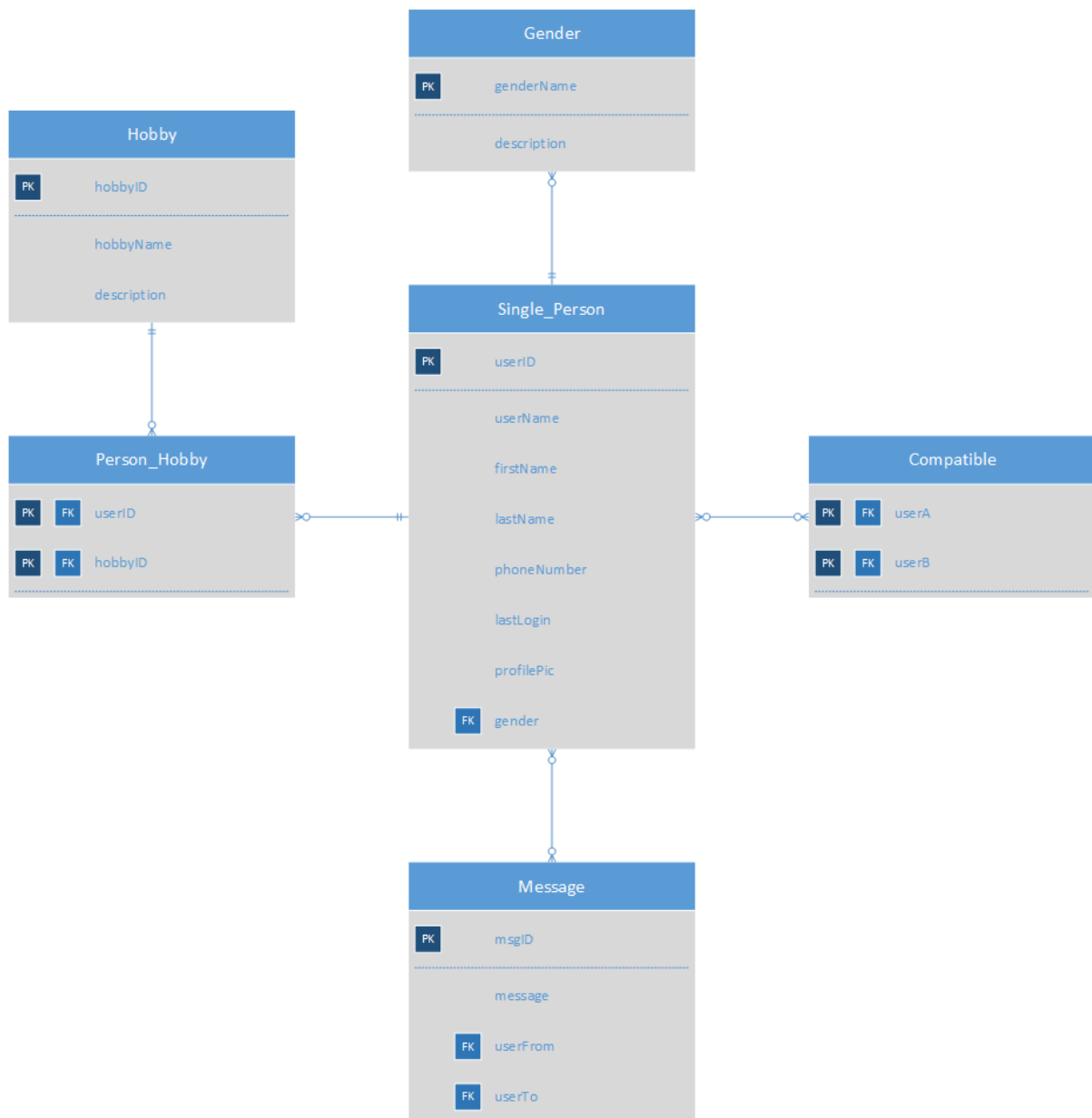


Figure 3 - Full ERD

This ERD will be developed further once the database design has been finalised. Please refer to the appropriate diagram later in this submission. A JPG image of this ERD will be submitted alongside this document.

b) Logical & Physical Design

Connolly and Begg (2015) defined a logical database design as the processing of developing a data model that specific to the enterprise but independent of any DBMS or any other physical constraints.

Hobby Table Design

The hobby table will store information relating to a hobby. As outlined in Table 3, the Manager entity consists of the following attributes:

1. ID;
2. Name; and;
3. Description.

Please refer to Table 3 for additional information on the attributes listed above. The following tables describe the meeting place entity in its entirety as per how it will be stored in the database.

Table 6 Properties of hobby table

Hobby					
Column Name	Data Type (length)	Key	Required	Default Value	Remark
hobbyID	Integer	Primary	Yes		Auto-generated by the DBMS.
hobbyName	Varchar	No	Yes	Null	Does not have to be unique.
description	Varchar	No	No	Null	Should be as brief as possible.

The SQL script to create the hobby table may look something like this

```
1 CREATE TABLE Hobby (  
2     hobbyID          INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT, /* PK */  
3     hobbyName        VARCHAR(64) ,  
4     hobbyDescription  VARCHAR(512)  
5 );
```

Figure 4 SQL Script to create Hobby Table

Single Person Table Design

The single person (user) table will store information relating to the manager of a single person. As outlined in Table 4, the Manager entity consists of the following attributes:

1. ID;
2. Username;
3. First name;
4. Surname;
5. Phone number;
6. Last date and time of login;
7. Gender; and;
8. Profile picture.

Please refer to Table 4 for additional information on the attributes listed above. The following tables describe the single person (user) entity in its entirety as per how it will be stored in the database.

Table 7 Properties of the single person table

Single Person					
Column Name	Data Type (length)	Key	Required	Default Value	Remark
userID	Integer	Primary	Yes		Auto-generated by the DBMS.
userName	Varchar	No	Yes	Null	Should be unique.
firstName	Varchar	No	Yes	Null	Does not have to be unique.
lastName	Varchar	No	No	Null	Does not have to be unique.
phoneNumber	Varchar	No	No	Null	Should be unique.
lastLogin	Date	No	No	Null	Date and time of the last login. Use a trigger to set.
profilePic	Blob	No	No	Null	Use a binary large object to store a picture of the user.

gender	Varchar	Foreign	No	Null	Foreign key to the gender table.
--------	---------	---------	----	------	----------------------------------

The SQL script that follows is a prototype create statement for the single person table.

```

1 CREATE TABLE SinglePerson (
2     userID          INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT, /* PK */
3     userName        VARCHAR(32) NOT NULL,
4     firstName       VARCHAR(32) NOT NULL,
5     lastName        VARCHAR(32),
6     phoneNumber     VARCHAR(10),
7     gender          VARCHAR(32),
8
9     FOREIGN KEY (gender) /* FK */
10    REFERENCES Gender(genderName)
11    ON UPDATE CASCADE ON DELETE CASCADE
12 );

```

Figure 5 SQL script to create single person table

Person Hobby Table Design

The person hobby table links single person entity and hobby entity together in a many-to-many relationship. The entity consists of the following attributes:

1. User ID; and;
2. Hobby ID.

The following tables describe the hobby person entity in its entirety as per how it will be stored in the database.

Table 8 Properties of person hobby linkage table

Person_Hobby					
Column Name	Data Type (length)	Key	Required	Default Value	Remark
userID	Integer	Primary Foreign	Yes	None	Surrogate key to the Single Person Table. Forms part of a composite primary key.
hobbyID	Integer	Primary Foreign	Yes	None	Surrogate key to the Hobby Table. Forms part of a composite primary key.

The SQL script that follows is a prototype create statement for the person hobby table.

```
1 CREATE TABLE Person_Hobby (  
2     userID      INTEGER NOT NULL, /* PK */  
3     hobbyID     INTEGER NOT NULL, /* PK */  
4  
5     PRIMARY KEY (userID, hobbyID),  
6  
7     FOREIGN KEY (userID) /* FK */  
8     REFERENCES SinglePerson(userID)  
9     ON UPDATE CASCADE ON DELETE CASCADE,  
10  
11    FOREIGN KEY (hobbyID) /* FK */  
12    REFERENCES Hobby(hobbyID)  
13    ON UPDATE CASCADE ON DELETE CASCADE  
14 );
```

Figure 6 SQL script to create person hobby table

As seen above, referential and entity integrity is maintained

Message Table Design

The message table will store information relating to the manager of a meeting place. As outlined in Table 5, the Manager entity consists of the following attributes:

1. ID;
2. Message;
3. From user; and;
4. To user.

Please refer to Table 5 for additional information on the attributes listed above. The following tables describe the message entity in its entirety as per how it will be stored in the database.

Table 9 Properties of message table

Message					
Column Name	Data Type (length)	Key	Required	Default Value	Remark
msgID	Integer	Primary	Yes		Auto-generated by the DBMS.
Message	Varchar	No	Yes	Null	The message. Should not be null.
userFrom	Integer	Foreign	Yes	None	Surrogate key. The primary key of the user who sent the message.
userTo	Integer	Foreign	Yes	None	Surrogate key. The primary key of the user that needs to receive the message.

The SQL script that follows is a prototype create statement for the message table.

```
1 CREATE TABLE Message (  
2     msgID      INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT, /* PK */  
3     message    VARCHAR(512),  
4     userFrom   INTEGER NOT NULL,  
5     userTo     INTEGER NOT NULL,  
6  
7     FOREIGN KEY (userFrom) /* FK */  
8     REFERENCES SinglePerson(userID)  
9     ON UPDATE CASCADE ON DELETE CASCADE,  
10  
11    FOREIGN KEY (userTo) /* FK */  
12    REFERENCES SinglePerson(userID)  
13    ON UPDATE CASCADE ON DELETE CASCADE  
14 );
```

Figure 7 SQL script to create message table

As seen above, referential and entity integrity is maintained

Compatibility Table Design

The compatibility (match) table will store information relating to the manager of a meeting place. As outlined in Table 6, the Manager entity consists of the following attributes:

1. User A; and;
2. User B.

Please refer to Table 6 for additional information on the attributes listed above. The following tables describe the compatibility (match) entity in its entirety as per how it will be stored in the database.

Table 10 Properties of compatibility table

Compatibility					
Column Name	Data Type (length)	Key	Required	Default Value	Remark
userA	Integer	Primary Foreign	Yes	None	Surrogate key to the Single Person Table. Forms part of a composite primary key.
userB	Integer	Primary Foreign	Yes	None	Surrogate key to the Single Person Table. Forms part of a composite primary key.

The SQL script that follows is a prototype create statement for the compatible table.

```
1 CREATE TABLE Compatible (  
2     userA    INTEGER NOT NULL, /* PK */  
3     userB    INTEGER NOT NULL, /* PK */  
4  
5     PRIMARY KEY (userA, userB),  
6  
7     FOREIGN KEY (userA) /* FK */  
8     REFERENCES SinglePerson(userID)  
9     ON UPDATE CASCADE ON DELETE CASCADE,  
10  
11    FOREIGN KEY (userB) /* FK */  
12    REFERENCES SinglePerson(userID)  
13    ON UPDATE CASCADE ON DELETE CASCADE  
14 );
```

Figure 8 SQL script to create compatible table

As seen above, referential and entity integrity is maintained

Gender Table Design

The gender table will store information relating to the manager of a meeting place. As outlined in Table 7, the Manager entity consists of the following attributes:

1. Name; and;
2. Description.

Please refer to Table 7 for additional information on the attributes listed above. The following tables describe the gender entity in its entirety as per how it will be stored in the database.

Table 11 Properties of gender table

Gender					
Column Name	Data Type (length)	Key	Required	Default Value	Remark
genderName	Varchar	Primary	Yes	Null	Needs to be unique (primary key for the table).
description	Varchar	No	Yes	Null	Should be as brief as possible.

The SQL script that follows is a prototype create statement for the gender table.

```
1 CREATE TABLE Gender (  
2     genderName    VARCHAR(32)    NOT NULL PRIMARY KEY, /* PK */  
3     description   VARCHAR(1024)  NOT NULL  
4 );
```

Figure 9 SQL script to create gender table

As seen above, referential and entity integrity is maintained

The ERD below is a graphical representation of the information described above. The ERD contains both attributes and datatypes.

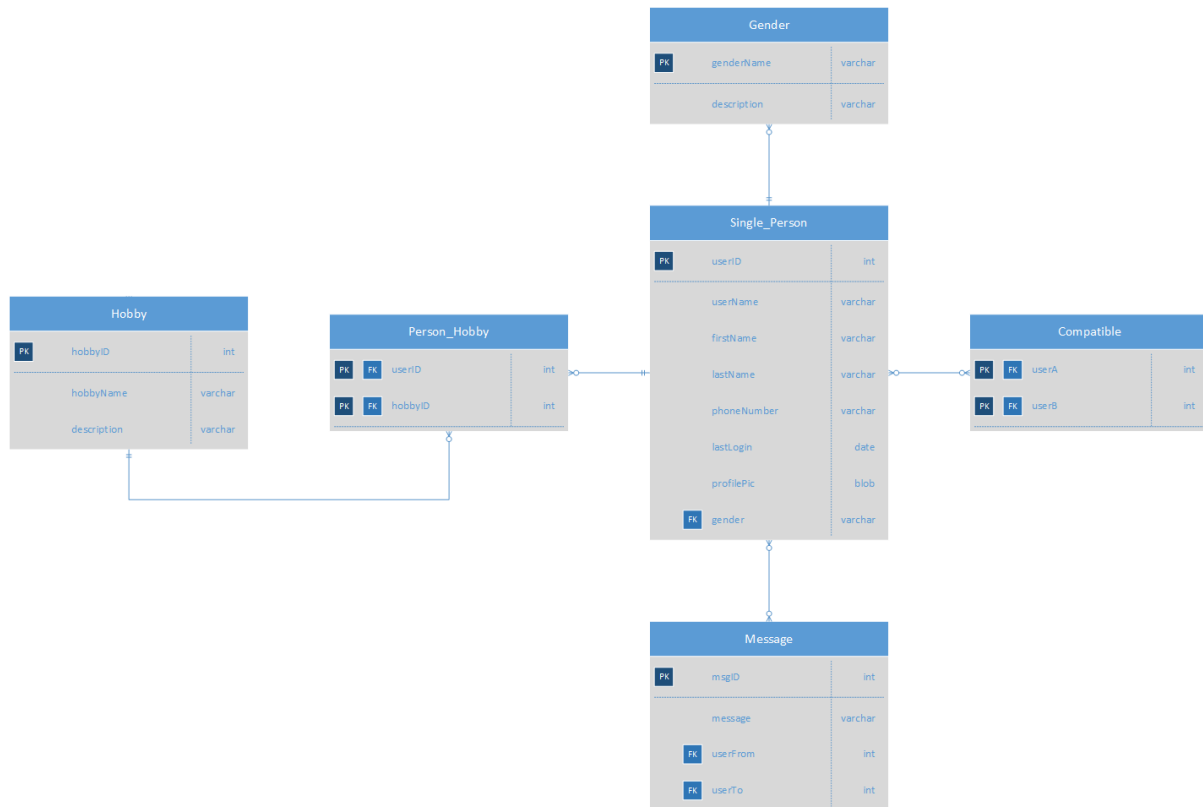


Figure 10 ERD with attributes & data types

The section that follows describes the creation of the of a fully functional database.

c) Fully Function Database

Please refer to the readme file handed in alongside this document. The final system will make use moderns design principles as suggested by Nielsen (1994), Valacich, et al. (2015), Bennet, et al. (2010) and Preece, et al. (2015). Modern best practices and modern object orientated programming concepts will be employed as suggested by Gaddis (2016), Deitel and Deitel (2012), Weisfeld (2013), Pretorius and Erasmus (2012) and Drake (2014).

All Graphical User Interfaces (GUI) have been made using the NetBeans Integrated Development Environment (IDE). The SQL statements below were used to create the database. It has been derived by examples given by Connolly & Begg (2015).

```
MariaDB [(none)]> CREATE DATABASE dating_database;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> SHOW DATABASES;
+-----+
| Database |
+-----+
| dating_database |
| information_schema |
| mysql |
| performance_schema |
| phpmyadmin |
+-----+
5 rows in set (0.00 sec)

MariaDB [(none)]> USE dating_database;
Database changed
MariaDB [dating database]>
```

Figure 11 Creating the database

As shown above, the statements executed correctly and the *dateing_database* has been selected. The project makes use of this database.

Create Statements

The create statements have been created by following examples presented by Connolly & Begg (2015).

```
MariaDB [dating_database]> CREATE TABLE Compatible (  
-> userA INTEGER NOT NULL, /* PK */  
-> userB INTEGER NOT NULL, /* PK */  
->  
-> PRIMARY KEY (userA, userB),  
->  
-> FOREIGN KEY (userA) /* FK */  
-> REFERENCES SinglePerson(userID)  
-> ON UPDATE CASCADE ON DELETE CASCADE,  
->  
-> FOREIGN KEY (userB) /* FK */  
-> REFERENCES SinglePerson(userID)  
-> ON UPDATE CASCADE ON DELETE CASCADE  
-> );  
Query OK, 0 rows affected (0.02 sec)
```

Figure 12 Creating the Compatible table

As indicated above the query executed successfully and the *Compatible* table has been created.

```
MariaDB [dating_database]> CREATE TABLE Gender (  
-> genderName VARCHAR(32) NOT NULL PRIMARY KEY, /* PK */  
-> description VARCHAR(1024) NOT NULL  
-> );  
Query OK, 0 rows affected (0.03 sec)
```

Figure 13 Creating the Gender table

As indicated above the query executed successfully and the *Gender* table has been created.

```
MariaDB [dating_database]> CREATE TABLE Hobby (  
-> hobbyID INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT, /* PK */  
-> hobbyName VARCHAR(64),  
-> hobbyDescription VARCHAR(512)  
-> );  
Query OK, 0 rows affected (0.02 sec)
```

Figure 14 Creating the Hobby table

As indicated above the query executed successfully and the *Hobby* table has been created.

```
MariaDB [dating_database]> CREATE TABLE Message (  
-> msgID INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT, /* PK */  
-> message VARCHAR(512),  
-> userFrom INTEGER NOT NULL,  
-> userTo INTEGER NOT NULL,  
->  
-> FOREIGN KEY (userFrom) /* FK */  
-> REFERENCES SinglePerson(userID)  
-> ON UPDATE CASCADE ON DELETE CASCADE,  
->  
-> FOREIGN KEY (userTo) /* FK */  
-> REFERENCES SinglePerson(userID)  
-> ON UPDATE CASCADE ON DELETE CASCADE  
-> );  
Query OK, 0 rows affected (0.02 sec)
```

Figure 15 Creating the Message table

As indicated above the query executed successfully and the *Message* table has been created.

```
MariaDB [dating_database]> CREATE TABLE Person_Hobby (  
-> userID INTEGER NOT NULL, /* PK */  
-> hobbyID INTEGER NOT NULL, /* PK */  
->  
-> PRIMARY KEY (userID, hobbyID),  
->  
-> FOREIGN KEY (userID) /* FK */  
-> REFERENCES SinglePerson(userID)  
-> ON UPDATE CASCADE ON DELETE CASCADE,  
->  
-> FOREIGN KEY (hobbyID) /* FK */  
-> REFERENCES Hobby(hobbyID)  
-> ON UPDATE CASCADE ON DELETE CASCADE  
-> );  
Query OK, 0 rows affected (0.02 sec)
```

Figure 16 Creating a table to link a person to the hobby

As indicated above the query executed successfully and the *Hobby* table has been created.

```
MariaDB [dating_database]> CREATE TABLE SinglePerson (  
-> userID INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT, /* PK */  
-> userName VARCHAR(32) NOT NULL,  
-> firstName VARCHAR(32) NOT NULL,  
-> lastName VARCHAR(32),  
-> phoneNumber VARCHAR(10),  
-> gender VARCHAR(32),  
->  
-> FOREIGN KEY (gender) /* FK */  
-> REFERENCES Gender(genderName)  
-> ON UPDATE CASCADE ON DELETE CASCADE  
-> );  
Query OK, 0 rows affected (0.03 sec)
```

Figure 17 Creating the user table

As indicated above the query executed successfully and the *Single Person* table has been created.

Stored Procedures

Match Users Stored Procedure

The SQL script that follows demonstrates how the *match_user* stored procedure was created. Stored procedures have been created by following examples presented by Connolly & Begg (2015).

```
1  USE `dateing_database`;  
2  DROP procedure IF EXISTS `match_users`;  
3  
4  DELIMITER $$  
5  USE `dateing_database` $$  
6  CREATE DEFINER=`root`@`localhost` PROCEDURE `match_users` (  
7      `userA` INTEGER,  
8      `userB` INTEGER  
9  )  
10 BEGIN  
11     INSERT INTO `dateing_database`.`compatible`  
12     (`userA`, `userB`) VALUES (`userA`, `userB`);  
13 END$$  
14  
15 DELIMITER ;
```

Figure 18 Match User stored Procedure

The *match_user* stored procedure is used to match two users together.

Add Hobby Stored Procedures

The SQL script that follows demonstrates how the *add_hobby* stored procedure was created.

```
1 • CREATE DEFINER='root'@'localhost' PROCEDURE `add_hobby` (  
2     `name`          VARCHAR(64),  
3     `description`   VARCHAR(64)  
4 )  
5 BEGIN  
6     INSERT INTO hobby (`hobbyName`, `hobbyDescription`) VALUES (`name`, `description`);  
7 END  
8
```

Figure 19 SQL script used to create hobby procedure

```
1  USE `dateing_database`;  
2  DROP procedure IF EXISTS `add_hobby`;  
3  
4  DELIMITER $$  
5  USE `dateing_database` $$  
6  CREATE PROCEDURE `add_hobby` (  
7      `name`          VARCHAR(64),  
8      `description`   VARCHAR(64)  
9  )  
10 BEGIN  
11     INSERT INTO hobby (`hobbyName`, `hobbyDescription`) VALUES (`name`,  
12     END$$  
13  
14 DELIMITER ;  
15
```

Figure 20 Add Hobby stored Procedure

The above query executed successfully and the *add_hobby* stored procedure was created.

Count Messages Stored Procedure

The count message stored procedure will be called from a trigger.

```
1 CREATE DEFINER=`root`@`localhost` PROCEDURE `cnt_msg`(id INTEGER)
2 BEGIN
3     SELECT COUNT(*) AS 'if exists' FROM message WHERE msgID = id;
4 END
```

Figure 21 count messages stored procedure

The count message will be used to determine whether a message already exists in the database. This is a form of data validation. Please refer to the trigger section of this document. The trigger

Triggers

This project will make use of triggers for database validation to ensure that the database remains consistent. Specifically, one trigger will determine whether user's phone number is only numbers, and another will determine whether a message exists within the database or not. Triggers have been created by following examples presented by Connolly & Begg (2015).

The trigger below validated a user's phone number.

```
1 CREATE DEFINER = CURRENT_USER TRIGGER `dateing_database`.`singleperson_BEFORE_INSERT`  
2 BEFORE INSERT ON `singleperson` FOR EACH ROW  
3 BEGIN  
4     IF (NEW.`phoneNumber` LIKE '%[^0-9]%') THEN  
5         SIGNAL SQLSTATE VALUE '45000'  
6         SET MESSAGE_TEXT = 'Please enter a phone number with only numbers.';  
7     END IF;  
8 END
```

Figure 22 Trigger to validate phone number (only numbers)

The trigger below guarantees that each entry into the message table has a unique ID. This has been done to further ensure entity integrity.

```
1 CREATE  
2 DEFINER=`root`@`localhost`  
3 TRIGGER `dateing_database`.`message_BEFORE_INSERT`  
4 BEFORE INSERT ON `dateing_database`.`message`  
5 FOR EACH ROW  
6 BEGIN  
7     IF (cnt_msg(NEW.`msgID`) > 1) THEN  
8         SIGNAL SQLSTATE VALUE '45000'  
9         SET MESSAGE_TEXT = 'That message already exist';  
10    END IF;  
11 END
```

Figure 23 Trigger to validate message ID

The SQL script was successfully executed, and the trigger was created. As seen above, this trigger makes use of a stored procedure.

Indices

Indices have been created by following examples presented by Connolly & Begg (2015). The following indices exist within the database.

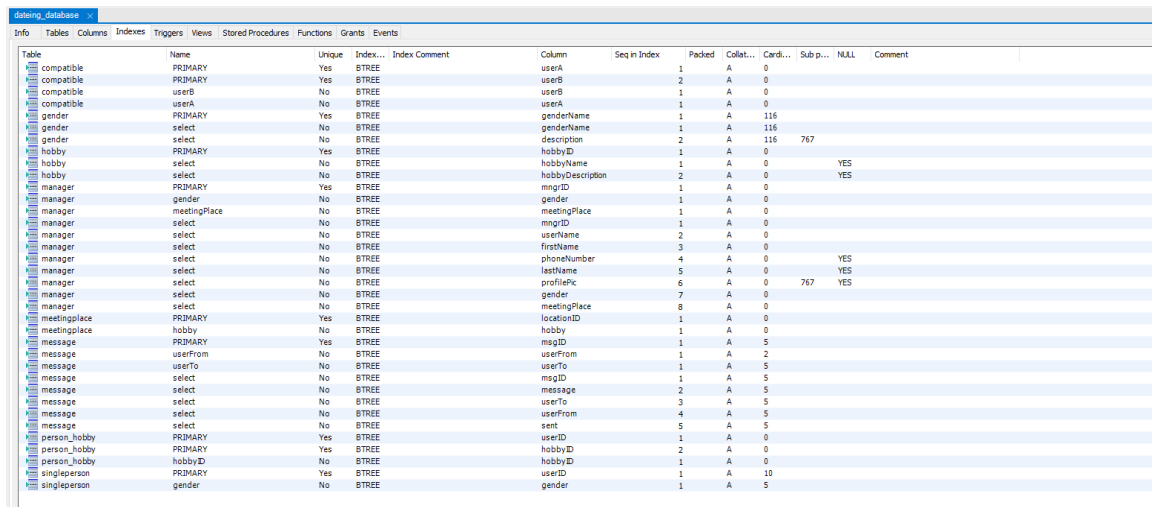


Table	Name	Unique	Index...	Index Comment	Column	Seq in Index	Packed	Collat...	Card...	Sub p...	NULL	Comment
compatible	PRIMARY	Yes	BTREE		userA	1	A		0			
compatible	PRIMARY	Yes	BTREE		userB	2	A		0			
compatible	userB	No	BTREE		userB	1	A		0			
compatible	userA	No	BTREE		userA	1	A		0			
gender	PRIMARY	Yes	BTREE		genderName	1	A		116			
gender	select	No	BTREE		genderName	1	A		116			
gender	select	No	BTREE		description	2	A		116	767		
hobby	PRIMARY	Yes	BTREE		hobbyID	1	A		0			
hobby	select	No	BTREE		hobbyName	1	A		0		YES	
hobby	select	No	BTREE		hobbyDescription	2	A		0		YES	
manager	PRIMARY	Yes	BTREE		mgrID	1	A		0			
manager	gender	No	BTREE		gender	1	A		0			
manager	meetingPlace	No	BTREE		meetingPlace	1	A		0			
manager	select	No	BTREE		mgrID	1	A		0			
manager	select	No	BTREE		userName	2	A		0			
manager	select	No	BTREE		firstName	3	A		0			
manager	select	No	BTREE		phoneNumber	4	A		0		YES	
manager	select	No	BTREE		lastName	5	A		0		YES	
manager	select	No	BTREE		profilePic	6	A		0	767	YES	
manager	select	No	BTREE		gender	7	A		0			
manager	select	No	BTREE		meetingPlace	8	A		0			
meetingplace	PRIMARY	Yes	BTREE		locationID	1	A		0			
meetingplace	hobby	No	BTREE		hobby	1	A		0			
message	PRIMARY	Yes	BTREE		msgID	1	A		5			
message	userFrom	No	BTREE		userFrom	1	A		2			
message	userTo	No	BTREE		userTo	1	A		5			
message	select	No	BTREE		msgID	1	A		5			
message	select	No	BTREE		message	2	A		5			
message	select	No	BTREE		userTo	3	A		5			
message	select	No	BTREE		userFrom	4	A		5			
message	select	No	BTREE		sent	5	A		5			
person_hobby	PRIMARY	Yes	BTREE		userID	1	A		0			
person_hobby	PRIMARY	Yes	BTREE		hobbyID	2	A		0			
person_hobby	hobbyID	No	BTREE		hobbyID	1	A		0			
singleperson	PRIMARY	Yes	BTREE		userID	1	A		10			
singleperson	gender	No	BTREE		gender	1	A		5			

Figure 24 Indices within the database

The subsections that follow display the creation of six developers created a table. These indices were created using Workbench.

Message Table Index

The following index was created on the messages table.

Index Name	Type	Index Columns			
PRIMARY	PRIMARY	Column	#	Order	Length
userFrom	INDEX	<input checked="" type="checkbox"/> msgID	1	ASC	
userTo	INDEX	<input checked="" type="checkbox"/> message	2	ASC	
select	INDEX	<input checked="" type="checkbox"/> userFrom	4	ASC	
		<input checked="" type="checkbox"/> userTo	3	ASC	
		<input checked="" type="checkbox"/> sent	5	ASC	

Figure 25 Message table index

The following query was created.

```
1 ALTER TABLE `dateing_database`.`message`  
2 ADD INDEX `select` (`msgID` ASC, `message` ASC, `userTo` ASC, `userFrom` ASC, `sent` ASC);
```

Figure 26 Query to create an index

The query above was successfully executed and an index on the message table was successfully created.

Gender Table Index

The following index was created on the gender table.

Index Name	Type	Index Columns			
PRIMARY	PRIMARY	Column	#	Order	Length
select	INDEX	<input checked="" type="checkbox"/> genderName	1	ASC	
		<input checked="" type="checkbox"/> description	2	ASC	

< >

Columns Indexes Foreign Keys Triggers Partitioning Options

Figure 27 Index on gender table

The following query was created.

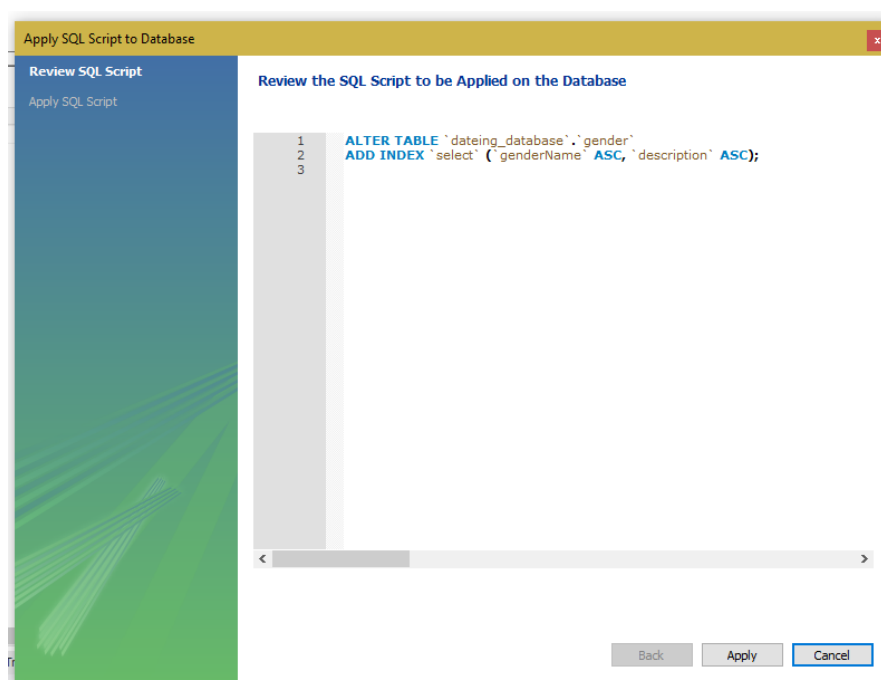


Figure 28 SQL Script used to create an index on gender table

The query above was successfully executed and an index on the gender table was successfully created.

Hobby Table Index

The following index was created on the hobby table.

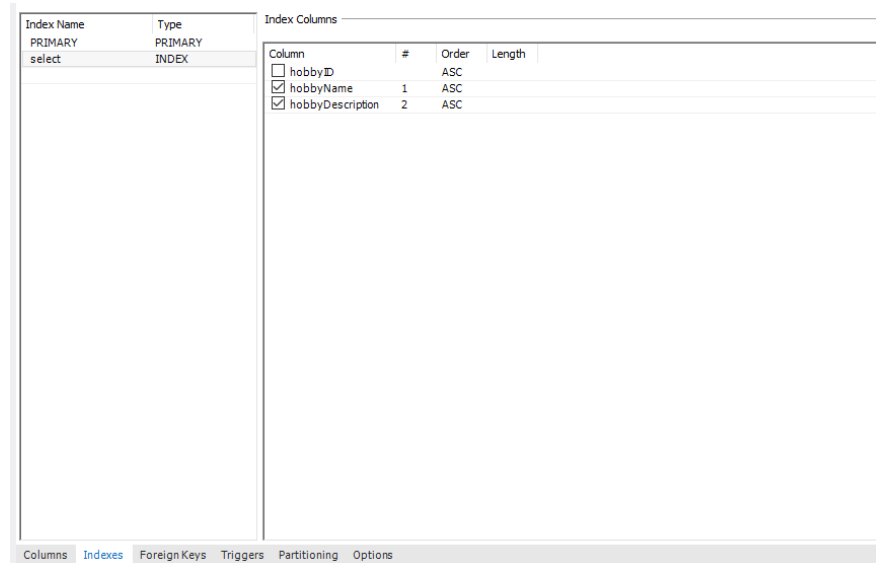


Figure 29 Index on hobby table

The following query was created.

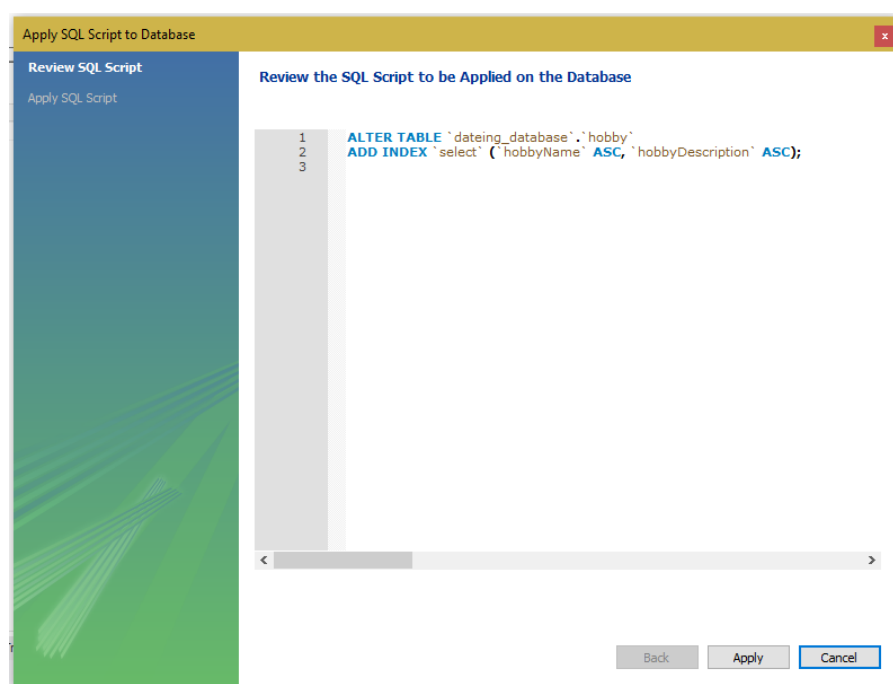


Figure 30 SQL script to create an index on hobby table

The query above was successfully executed and an index on the hobby table was successfully created.

Single Person Table Index

The single person table had the following default index.

Index Name	Type	Index Columns			
PRIMARY	PRIMARY	Column	#	Order	Length
gender	INDEX	<input type="checkbox"/> userID		ASC	
		<input type="checkbox"/> userName		ASC	
		<input type="checkbox"/> passwd		ASC	
		<input type="checkbox"/> firstName		ASC	
		<input type="checkbox"/> lastName		ASC	
		<input type="checkbox"/> phoneNumber		ASC	
		<input type="checkbox"/> profilePic		ASC	
		<input checked="" type="checkbox"/> gender	1	ASC	

ColumnsIndexesForeign KeysTriggersPartitioningOptions

Figure 31 Index on single person table

Compatible Table Index

The single compatible table had the following default index.

Index Name	Type	Index Columns			
PRIMARY	PRIMARY	Column	#	Order	Length
userB	INDEX	<input type="checkbox"/> userA		ASC	
userA	INDEX	<input type="checkbox"/> userB		ASC	

ColumnsIndexesForeign KeysTriggersPartitioningOptions

Figure 32 Index on compatible table

Views

Database views are a virtual representation of a table (Connolly & Begg, 2015). The correct usage of view can help improve the security of a database (Connolly & Begg, 2015). Views have been created by following examples presented by Connolly & Begg (2015).

Get Messages View

The *get_messages* view will return specific details of a message.

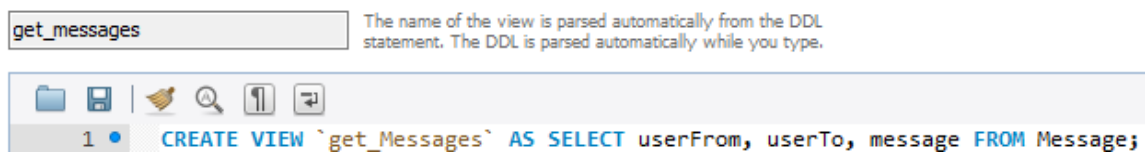


Figure 33 Creating a *get_message* view

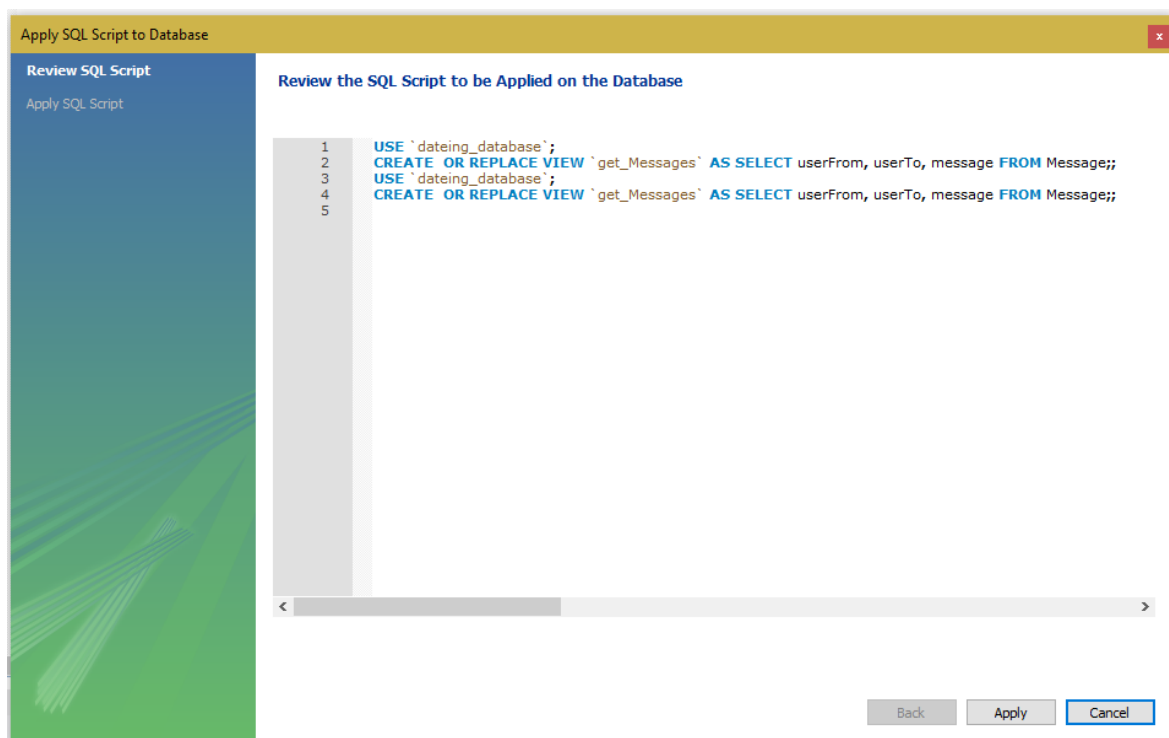


Figure 34 Creating a *get_messages* view

```
1 • CREATE
2     ALGORITHM = UNDEFINED
3     DEFINER = `root`@`localhost`
4     SQL SECURITY DEFINER
5 VIEW `dateing_database`.`get_messages` AS
6     SELECT
7         `dateing_database`.`message`.`userFrom` AS `userFrom`,
8         `dateing_database`.`message`.`userTo` AS `userTo`,
9         `dateing_database`.`message`.`message` AS `message`
10    FROM
11        `dateing_database`.`message`
```

Figure 35 SQL Script used to create a get_messages view

The query above was successfully executed and an index on the view was successfully created

Get hobbies View

The `get_hobbies` view will return distinct hobbies stored within the database.

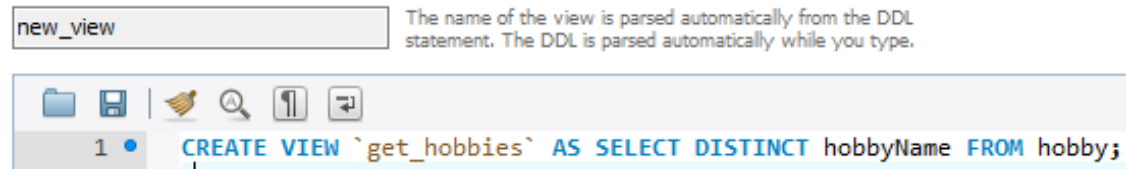


Figure 36 Creating get a view to get hobbies

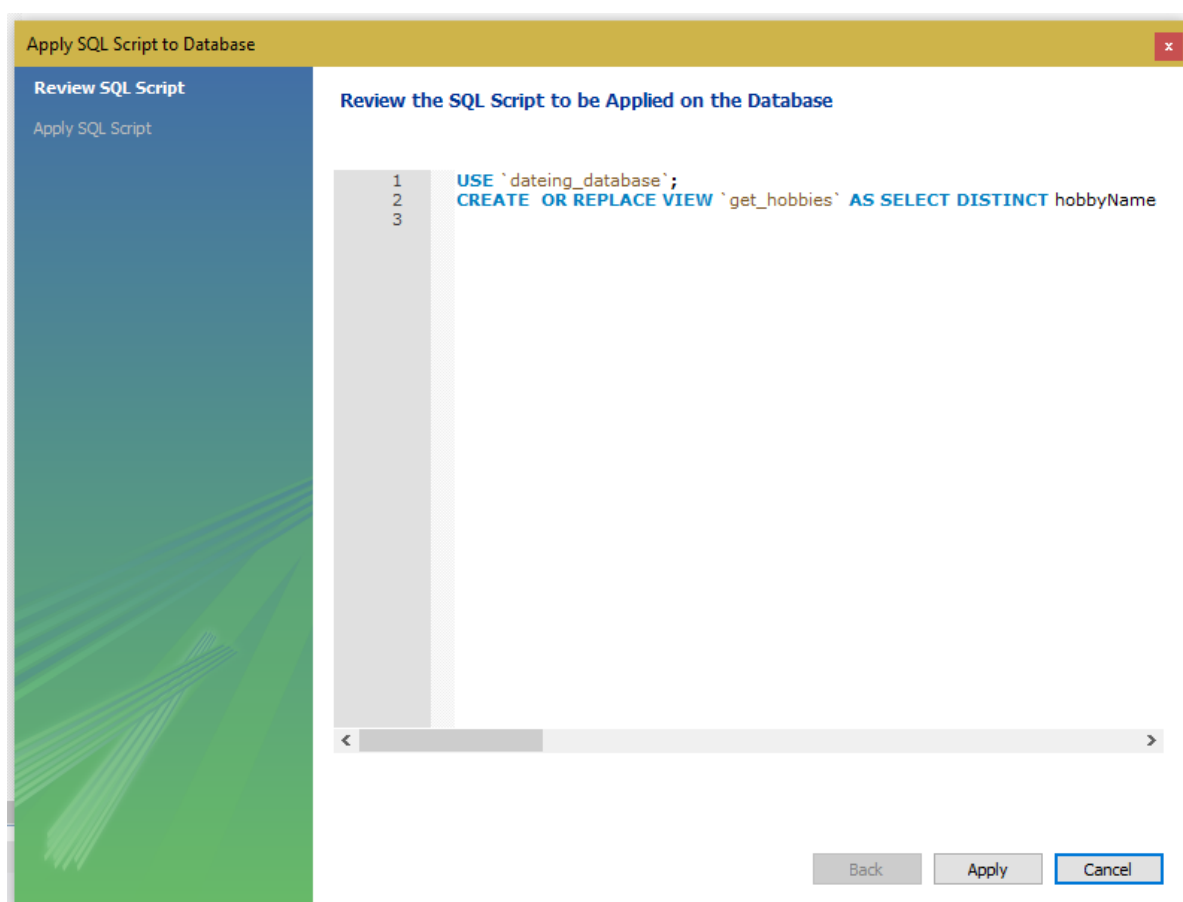


Figure 37 Creating get hobbies view


```
1 ● CREATE
2     ALGORITHM = UNDEFINED
3     DEFINER = `root`@`localhost`
4     SQL SECURITY DEFINER
5 VIEW `dateing_database`.`get_hobbies` AS
6     SELECT DISTINCT
7         `dateing_database`.`hobby`.`hobbyName` AS `hobbyName`
8     FROM
9         `dateing_database`.`hobby`
10
```

Figure 38 SQL script used to create get hobbies view

The query above was successfully executed and an index on the view was successfully created.

Final ERD

From here the final ERD can be generated. The ERD below was generated using Workbench.

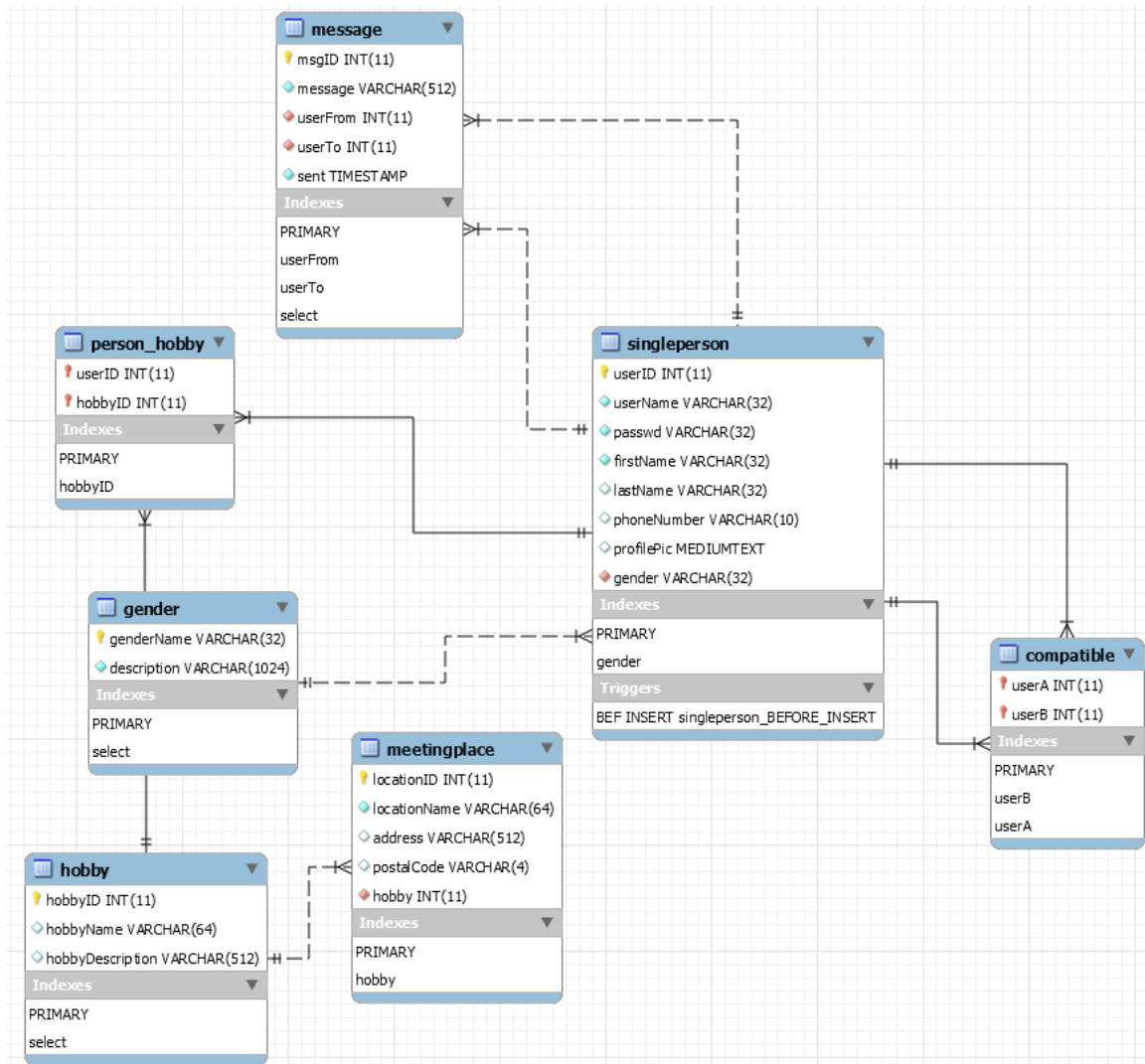


Figure 39 ERD generated by Workbench

The ERD that follows is a complete ERD for the entire system. A JPG image of this ERD will be submitted alongside this document.

Prototypes

The screenshot that follows is are prototype user interfaces. All interfaces were created using the NetBeans IDE. Please refer to the captions on each screenshot to determine what it is. All user interfaces make use of design principles suggested by Rogers, et al. (2011) and Nielsen (1994).

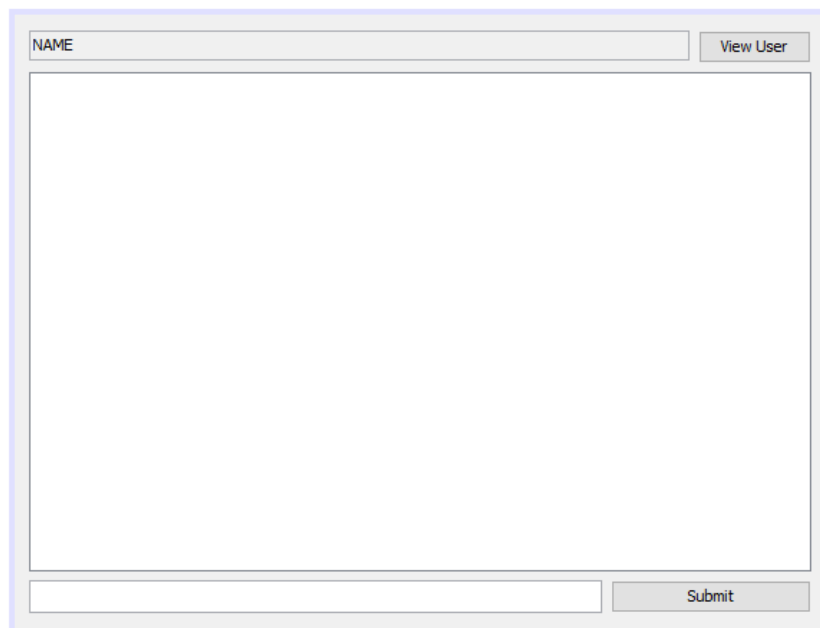


Figure 40 Prototype Message Panel

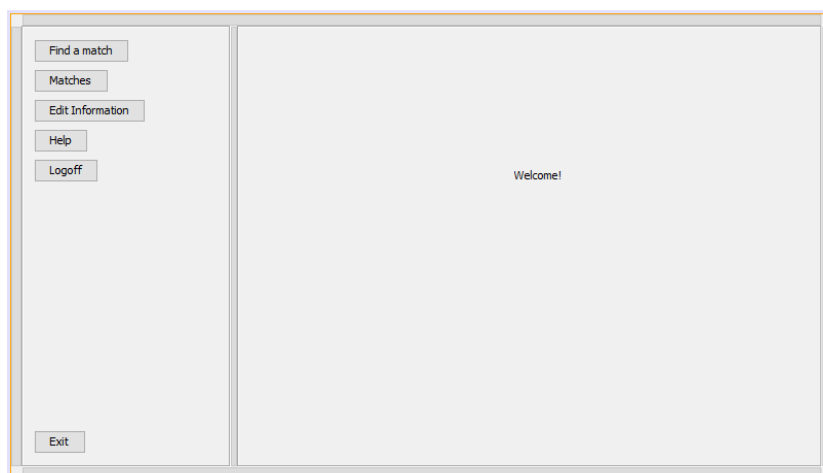


Figure 41 Prototype Main Frame

Username

First Name

Last Name

Phone Number

Gender

jLabel5

Figure 42 Prototype Information Editor

Register as User Register as Manager

Username

First Name

Last Name

Phone Number

Gender

Picture

Figure 43 Prototype Registration

Code Snippets

The Java code below connects the Java application to the SQL database. Only one creation is opened. The connection is closed when the main frame of the application is disposed of.

```
1  /**
2   * Creates database connection.
3   */
4  public static void init() {
5      try {
6          Class.forName("com.mysql.cj.jdbc.Driver");
7          CNCTN = DriverManager.getConnection(
8              "jdbc:mysql://localhost:3306/dateing_database", "root", "");
9      } catch (ClassNotFoundException | SQLException ex) {
10         ex.printStackTrace(System.err);
11     }
12 }
13
14 public static Statement getStatement() throws SQLException {
15     return Database.CNCTN.createStatement();
16 }
17
18 /**
19 * Closes database connection.
20 */
21 public static void dispose() {
22     try {
23         Database.CNCTN.close();
24     } catch (SQLException sqle) {
25         Logger.getLogger(Database.class.getName()).log(Level.SEVERE, null, sqle);
26     }
27 }
```

Figure 44 Java database connection

The Java code below updates a user's information. It makes use of the database connection.

```
1  public void update(Component parent) {
2      String q = String.format("UPDATE `dateing_database`.`singleperson` \n"
3          + "SET \n"
4          + "`userName` = '%s', \n"
5          + "`passwd` = '%s', \n"
6          + "`firstName` = '%s', \n"
7          + "`lastName` = '%s', \n"
8          + "`phoneNumber` = '%s', \n"
9          + "`profilePic` = '%s', \n"
10         + "`gender` = '%s' \n"
11         + "WHERE `userID` = %d;",
12         userName, passwd, firstName, lastName, phoneNumber, profilePic, gender, userID);
13
14     try {
15         Statement stmt = Database.getStatement();
16
17         Toolkit.getDefaultToolkit().beep();
18         if (!stmt.execute(q)) {
19             JOptionPane.showMessageDialog(parent, "Saved Sucessully", "Saved",
20                 JOptionPane.INFORMATION_MESSAGE);
21         } else {
22             JOptionPane.showMessageDialog(parent, "Failed to Saved Sucessully", "Failed",
23                 JOptionPane.ERROR_MESSAGE);
24         }
25
26         stmt.close();
27     } catch (SQLException sqle) {
28         sqle.printStackTrace(System.err);
29     }
30 }
31 }
```

Figure 45 Java code to update the database

The Java code below inserts a new user into the database. The code is run when a user registers in the system. The code makes use of the database connection.

```

1 public void insert(Component parent) {
2     String q = String.format("INSERT INTO `dateing_database`.`singleperson` (\n"
3         + "userName`, `passwd`, `firstName`, `lastName`, `phoneNumber`, \"
4         + "`profilePic`, `gender`) VALUES %s,%s,%s,%s,%s,%s,%s);",
5         getUser(), getPasswd(), getFirstName(), getLastName(), getPhoneNumber(),
6         profilePic, gender);
7
8     try {
9         Statement stmt = Database.getStatement();
10
11         if (!stmt.execute(q)) {
12             JOptionPane.showMessageDialog(parent, "Insert Sucessul", "Inserted",
13                 JOptionPane.INFORMATION_MESSAGE);
14         } else {
15             JOptionPane.showMessageDialog(parent, "Failed to Insert Sucessully", "Failed",
16                 JOptionPane.ERROR_MESSAGE);
17         }
18
19         stmt.close();
20     } catch (SQLException sqle) {
21         sqle.printStackTrace(System.err);
22     }
23 }

```

Figure 46 Java code to insert into the database

```

1 String q = String.format("DELETE FROM `dateing_database`.`hobby`\n"
2     + "WHERE hobbyName = '%s'", cmbHobbies.getSelectedItem().toString());
3
4 System.out.println(q);
5
6 try {
7     Statement stmt = Database.getStatement();
8
9     if (!stmt.execute(q)) {
10         JOptionPane.showMessageDialog(parent, "Removed Sucessul", "Removed",
11             JOptionPane.INFORMATION_MESSAGE);
12     } else {
13         Toolkit.getDefaultToolkit().beep();
14         JOptionPane.showMessageDialog(parent, "Failed to remove Sucessully", "Failed",
15             JOptionPane.ERROR_MESSAGE);
16     }
17
18     stmt.close();
19 } catch (SQLException sqle) {
20     sqle.printStackTrace(System.err);
21 }
22
23 this.parent.dispose();
24 FrameMain main = new FrameMain(parent.getUser(), parent);
25 main.changePannel(new PnlEditInformation(main));
26 main.setVisible(true);
27

```

Figure 47 Java code to delete from the database

```

1 String name = JOptionPane.showInputDialog(parent, "What is the name of "
2     + "your hobby?", "What is the name?", JOptionPane.QUESTION_MESSAGE);
3 String description = JOptionPane.showInputDialog(parent, "What is the description of
4     + "your hobby?", "What is the description?", JOptionPane.QUESTION_MESSAGE);
5
6 String q = String.format("call add_hobby('%s', '%s')", name, description);
7
8 System.out.println(q);
9
10 try {
11     Statement stmt = Database.getStatement();
12
13     if (!stmt.execute(q)) {
14         JOptionPane.showMessageDialog(parent, "Insert Sucessul", "Added",
15             JOptionPane.INFORMATION_MESSAGE);
16     } else {
17         Toolkit.getDefaultToolkit().beep();
18         JOptionPane.showMessageDialog(parent, "Failed to add Sucessully", "Failed",
19             JOptionPane.ERROR_MESSAGE);
20     }
21
22     stmt.close();
23 } catch (SQLException sqle) {
24     sqle.printStackTrace(System.err);
25 }
26
27 this.cmbHobbies.addItem(name);

```

Figure 48 Java code to call stored procedures

Final Interfaces

The main JFrame will act as the main/parent frame for the entire project. The main method will be held in the class named Main.java.

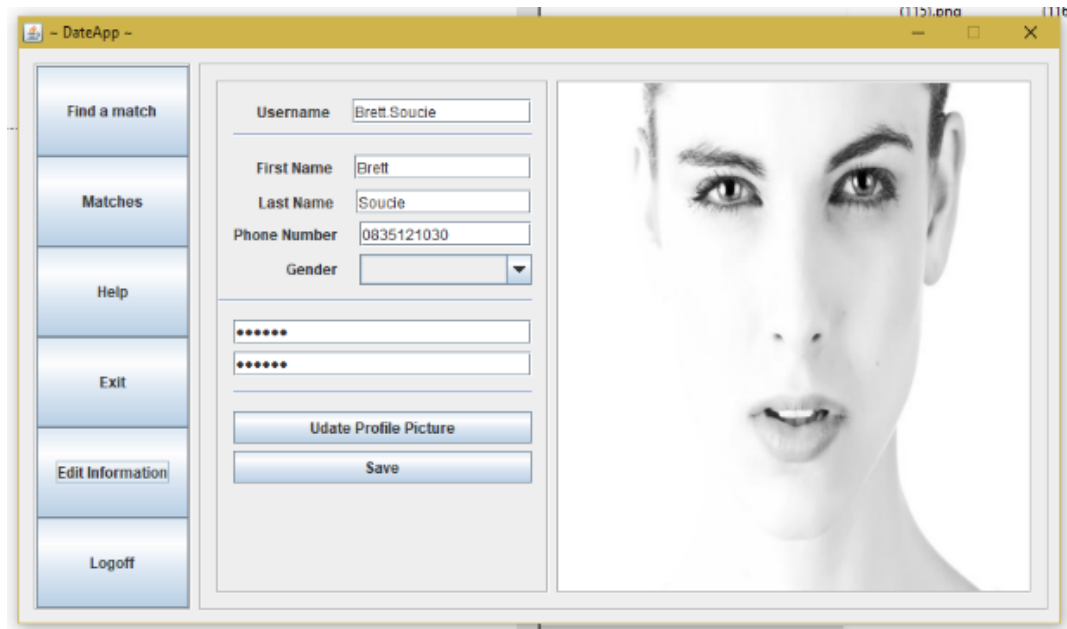


Figure 49 Edit information panel

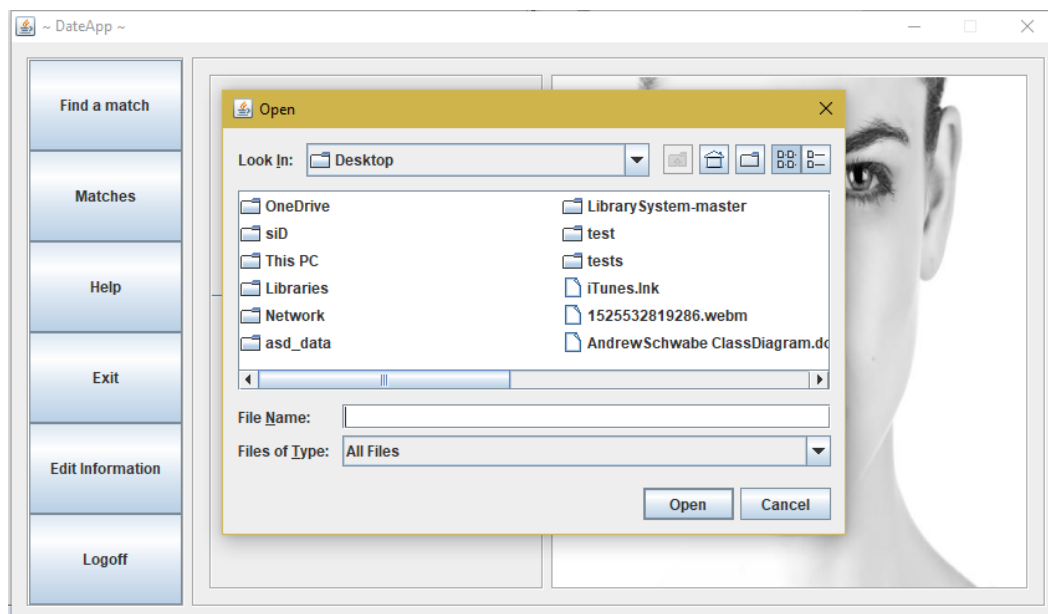


Figure 50 Select profile picture dialogue

The view user panel shows all relevant information about a user.

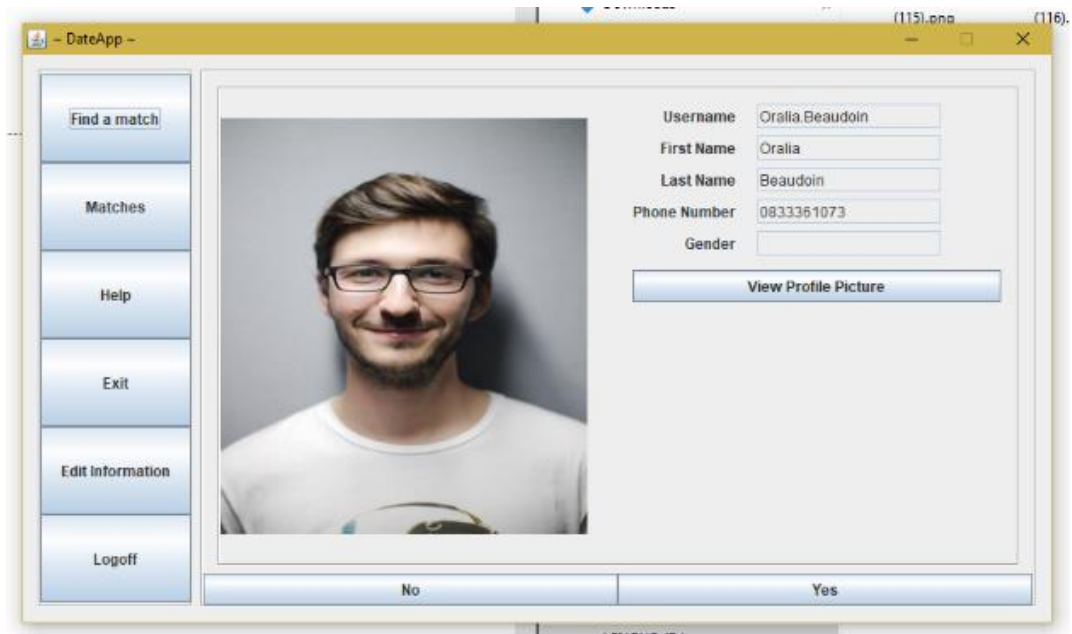


Figure 51 View user details panel

The view image dialogue is used to view a user's profile picture. It contains a JLabel with no a blank string as text and an image icon set to the width of the dialogue.

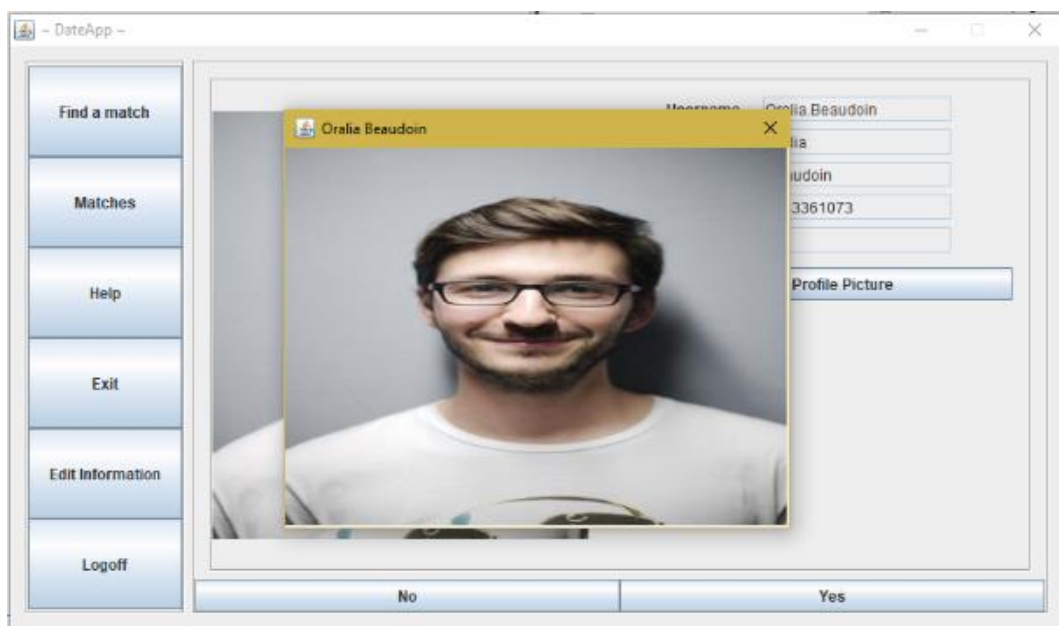


Figure 52 View image dialogue

The matches panel show all other users that the current user has matched with.



Figure 53 Matches panel

The messages panel facilitates messages between two users. The messages panel makes use of a database trigger.

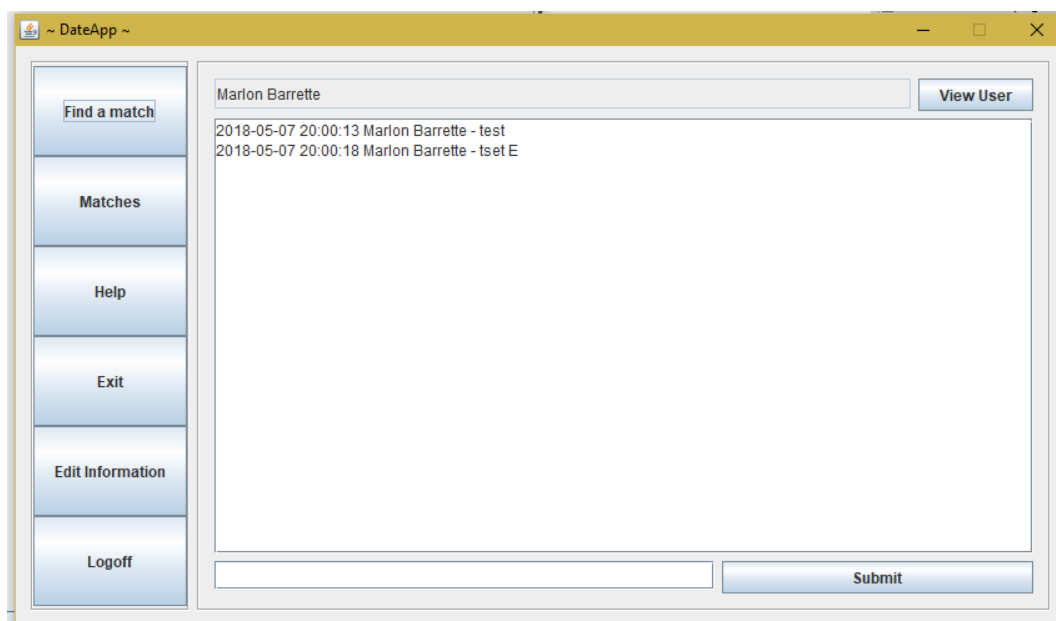


Figure 54 Message panel

Users will be asked to confirm when logging off the application.

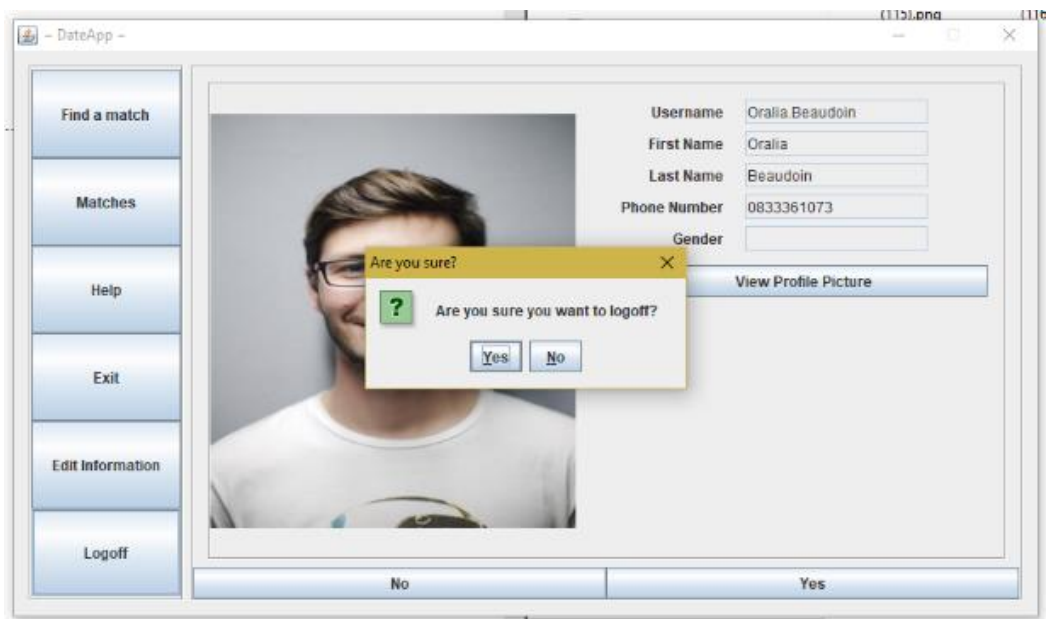


Figure 55 Logoff dialogue

Users will be asked to confirm when closing the application.

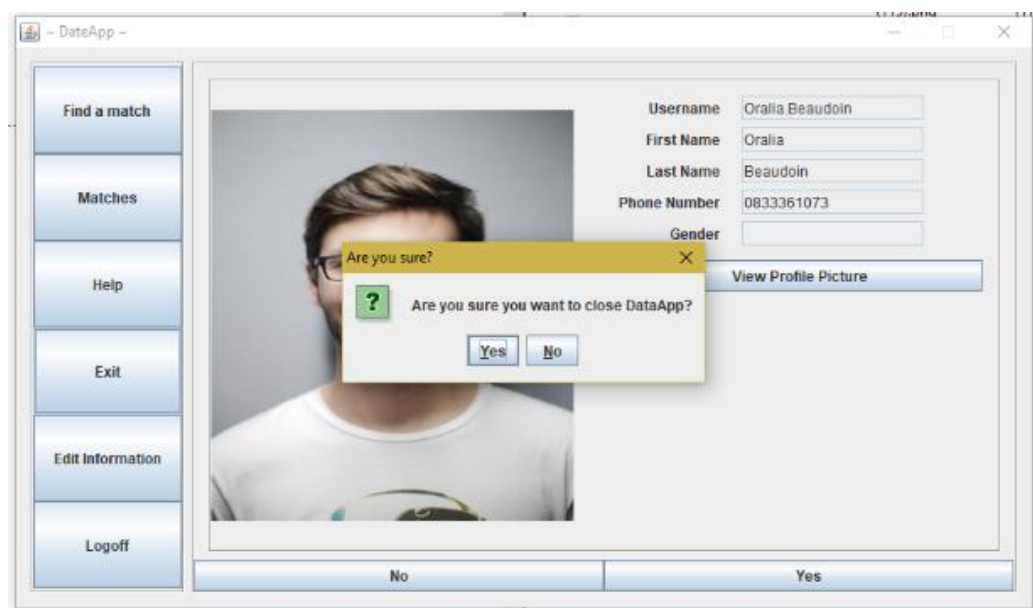


Figure 56 Exit dialogue

As indicated above, the program always confirms the user's intent when exiting or logging of the program.

1.2 Roles & Privileges

As per the requirements given, a minimum of two users and two roles will be created.

a) Roles

As shown in the figure below, the system contains three users and five roles. These users and roles are further defined in the section that follows.

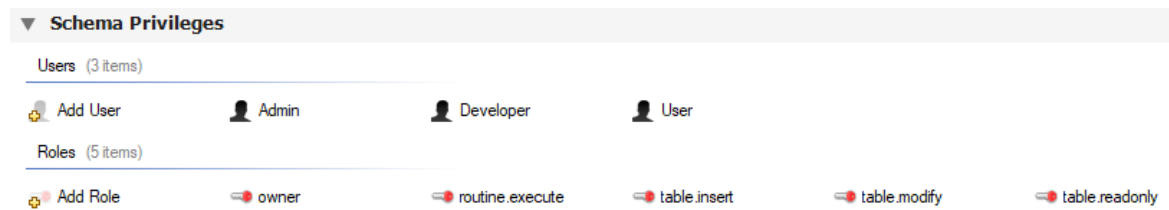


Figure 57 Users and Roles

Each user will be given a different role, as indicated in the table below.

Table 12 Users & Roles

User	Role
Admin	owner
Developer	table.modify
User	table.insert

Each user type will interact with the system differently. The table that follows further describes what privileges are associated with each role.


The table that follows indicates the privilege each role has been granted.

Table 13 Table of Roles

Roles and Privileges												
	Create	Drop	Grant Option	References	Event	Lock Tables	Insert	Select	Update	Trigger	Delete	Execute
Owner	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Routine Execute												✓
Table Insert							✓	✓	✓			
Table Modify								✓	✓		✓	
Table Read-only								✓				

b) Global & Schema Privileges

Privileges are used to limit what users can do on a database (Connolly & Begg, 2015). This is done to improve database security (Connolly & Begg, 2015).



xampp
Users and Privileges


User Accounts

User	From Host
(!) <anonymous>	%
<anonymous>	localhost
Admin	%
Developer	%
User	%
pma	localhost
root	localhost
root	127.0.0.1
root	::1
newuser	%

Figure 58 All users on the database

Developer Privileges

The screenshots that follow are of the creation of the developer user. The developer user was used in the creation of the project.



xampp
Users and Privileges

User Accounts

User	From Host
(!) <anonymous>	%
<anonymous>	localhost
Admin	%
Developer	%
User	%
pma	localhost
root	localhost
root	127.0.0.1
root	::1
newuser	%

Details for account newuser@%

Login Account Limits Administrative Roles Schema Privileges

Login Name: You may create multiple accounts with the same name to connect from different hosts.

Authentication Type: For the standard password and/or host based authentication, select 'Standard'.

Limit to Hosts Matching: % and _ wildcards may be used

Password: Type a password to reset it.

Consider using a password with 8 or more characters with mixed case letters, numbers and punctuation marks.

Confirm Password: Enter password again to confirm.

User Accounts

User	From Host
(!) <anonymous>	%
<anonymous>	localhost
Admin	%
Developer	%
User	%
pma	localhost
root	localhost
root	127.0.0.1
root	::1
newuser	%

Details for account newuser@%

Login Account Limits Administrative Roles Schema Privileges

Max. Queries: 0 Number of queries the account can execute within one hour.

Max. Updates: 0 Number of updates the account can execute within one hour.

Max. Connections: 0 The number of times the account can connect to the server per hour.

Concurrent Connections: 0 The number of simultaneous connections to the server the account can have.

Figure 59 Details of the developer user

User Accounts

User	From Host
(!) <anonymous>	%
<anonymous>	localhost
Admin	%
Developer	%
User	%
pma	localhost
root	localhost
root	127.0.0.1
root	::1
newuser	%

Details for account newuser@%


Login Account Limits Administrative Roles Schema Privileges

Role	Description
<input checked="" type="checkbox"/> DBA	grants the rights to perform all tasks
<input checked="" type="checkbox"/> MaintenanceAdmin	grants rights needed to maintain server
<input checked="" type="checkbox"/> ProcessAdmin	rights needed to assess, monitor, and kill any user proce...
<input checked="" type="checkbox"/> UserAdmin	grants rights to create users logins and reset passwords
<input checked="" type="checkbox"/> SecurityAdmin	rights to manage logins and grant and revoke server an...
<input checked="" type="checkbox"/> MonitorAdmin	minimum set of rights needed to monitor server
<input checked="" type="checkbox"/> DBManager	grants full rights on all databases
<input checked="" type="checkbox"/> DBDesigner	rights to create and reverse engineer any database sche...
<input checked="" type="checkbox"/> ReplicationAdmin	rights needed to setup and manage replication
<input checked="" type="checkbox"/> BackupAdmin	minimal rights needed to backup any database

Figure 60 Details of developer user

Global Privileges	
<input checked="" type="checkbox"/>	ALTER
<input checked="" type="checkbox"/>	ALTER ROUTINE
<input checked="" type="checkbox"/>	CREATE
<input checked="" type="checkbox"/>	CREATE ROUTINE
<input checked="" type="checkbox"/>	CREATE TABLESPACE
<input checked="" type="checkbox"/>	CREATE TEMPORARY TABLES
<input checked="" type="checkbox"/>	CREATE USER
<input checked="" type="checkbox"/>	CREATE VIEW
<input checked="" type="checkbox"/>	DELETE
<input checked="" type="checkbox"/>	DROP
<input checked="" type="checkbox"/>	EVENT
<input checked="" type="checkbox"/>	EXECUTE
<input checked="" type="checkbox"/>	FILE
<input checked="" type="checkbox"/>	GRANT OPTION
<input checked="" type="checkbox"/>	INDEX
<input checked="" type="checkbox"/>	INSERT
<input checked="" type="checkbox"/>	LOCK TABLES
<input checked="" type="checkbox"/>	PROCESS
<input checked="" type="checkbox"/>	REFERENCES
<input checked="" type="checkbox"/>	RELOAD
<input checked="" type="checkbox"/>	REPLICATION CLIENT
<input checked="" type="checkbox"/>	REPLICATION SLAVE
<input checked="" type="checkbox"/>	SELECT
<input checked="" type="checkbox"/>	SHOW DATABASES
<input checked="" type="checkbox"/>	SHOW VIEW
<input checked="" type="checkbox"/>	SHUTDOWN
<input checked="" type="checkbox"/>	SUPER
<input checked="" type="checkbox"/>	TRIGGER
<input checked="" type="checkbox"/>	UPDATE

Figure 61 Global privileges of the developer user


MySQL

Users and Privileges

User Accounts

User	From Host
{ } <anonymous>	%
<anonymous>	localhost
Admin	%
Developer	%
User	%
pma	localhost
root	localhost
root	127.0.0.1
root	:::
newuser	%

Details for account newuser@%

Login Account Limits Administrative Roles Schema Privileges

Schema	Privileges
%	ALTER, ALTER ROUTINE, CREATE, CREATE ROUTINE, CREATE TEMPORARY TABLES, CREATE VIEW, DELETE, DROP, EVENT, EXECUTE, INDEX, INSERT, LO...

Schema and host fields may use % and _ wildcards.
The server will match specific entries before wildcarded ones.

The user 'newuser'@'%' will have the following access rights to any schema:

Object Rights

- ☒ SELECT
- ☒ INSERT
- ☒ UPDATE
- ☒ DELETE
- ☒ EXECUTE
- ☒ SHOW VIEW

DOL Rights

- ☒ CREATE
- ☒ ALTER
- ☒ REFERENCES
- ☒ INDEX
- ☒ CREATE VIEW
- ☒ CREATE ROUTINE
- ☒ ALTER ROUTINE
- ☒ EVENT
- ☒ DROP
- ☒ TRIGGER

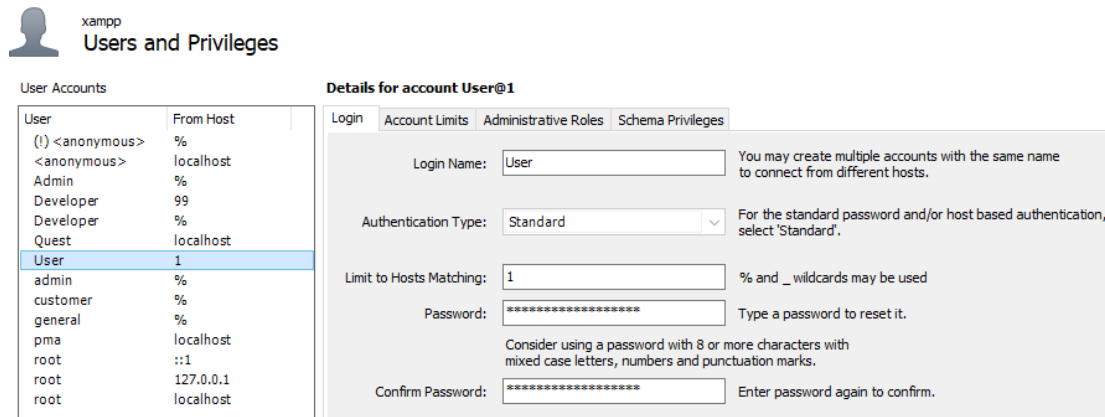
Other Rights

- ☐ GRANT OPTION
- ☒ CREATE TEMPORARY TABLES
- ☒ LOCK TABLES

Figure 62 Privileges of the developer user

User Privileges

The screenshots that follow are of the creation of the generic user. The developer user was used in the creation of the project.



The screenshot shows the 'Users and Privileges' window with the 'Login' tab selected for 'User@1'. The 'User Accounts' table on the left lists various users, with 'User' highlighted. The 'Login' tab on the right contains the following fields:

User	From Host
(!) <anonymous>	%
<anonymous>	localhost
Admin	%
Developer	99
Developer	%
Quest	localhost
User	1
admin	%
customer	%
general	%
pma	localhost
root	::1
root	127.0.0.1
root	localhost

Details for account User@1

Login | Account Limits | Administrative Roles | Schema Privileges

Login Name: You may create multiple accounts with the same name to connect from different hosts.

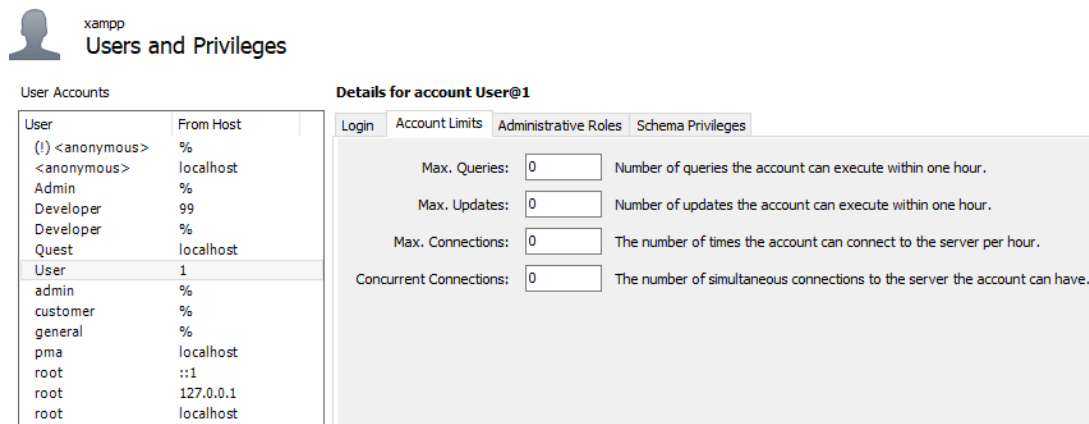
Authentication Type: For the standard password and/or host based authentication, select 'Standard'.

Limit to Hosts Matching: % and _ wildcards may be used

Password: Type a password to reset it.

Confirm Password: Enter password again to confirm.

Figure 63 Login details of a generic user



The screenshot shows the 'Users and Privileges' window with the 'Account Limits' tab selected for 'User@1'. The 'User Accounts' table on the left is the same as in Figure 63. The 'Account Limits' tab on the right contains the following fields:

User	From Host
(!) <anonymous>	%
<anonymous>	localhost
Admin	%
Developer	99
Developer	%
Quest	localhost
User	1
admin	%
customer	%
general	%
pma	localhost
root	::1
root	127.0.0.1
root	localhost

Details for account User@1

Login | **Account Limits** | Administrative Roles | Schema Privileges

Max. Queries: Number of queries the account can execute within one hour.

Max. Updates: Number of updates the account can execute within one hour.

Max. Connections: The number of times the account can connect to the server per hour.

Concurrent Connections: The number of simultaneous connections to the server the account can have.

Figure 64 Account limits for a generic user

User Accounts

User	From Host
(!) <anonymous>	%
<anonymous>	localhost
Admin	%
Developer	99
Developer	%
Quest	localhost
User	1
admin	%
customer	%
general	%
pma	localhost
root	:::1
root	127.0.0.1
root	localhost


Details for account User@1

Login	Account Limits	Administrative Roles	Schema Privileges
Role	Description		
<input type="checkbox"/> DBA	grants the rights to perform all tasks		
<input type="checkbox"/> MaintenanceAdmin	grants rights needed to maintain server		
<input type="checkbox"/> ProcessAdmin	rights needed to assess, monitor, and kill any user proce...		
<input type="checkbox"/> UserAdmin	grants rights to create users logins and reset passwords		
<input type="checkbox"/> SecurityAdmin	rights to manage logins and grant and revoke server an...		
<input type="checkbox"/> MonitorAdmin	minimum set of rights needed to monitor server		
<input checked="" type="checkbox"/> DBManager	grants full rights on all databases		
<input checked="" type="checkbox"/> DBDesigner	rights to create and reverse engineer any database sche...		
<input type="checkbox"/> ReplicationAdmin	rights needed to setup and manage replication		
<input checked="" type="checkbox"/> BackupAdmin	minimal rights needed to backup any database		
<input type="checkbox"/> Custom	custom role		

Figure 65 Roles for a generic user

Global Privileges
<input checked="" type="checkbox"/> ALTER
<input checked="" type="checkbox"/> ALTER ROUTINE
<input checked="" type="checkbox"/> CREATE
<input checked="" type="checkbox"/> CREATE ROUTINE
<input type="checkbox"/> CREATE TABLESPACE
<input checked="" type="checkbox"/> CREATE TEMPORARY TABLES
<input type="checkbox"/> CREATE USER
<input checked="" type="checkbox"/> CREATE VIEW
<input checked="" type="checkbox"/> DELETE
<input checked="" type="checkbox"/> DROP
<input checked="" type="checkbox"/> EVENT
<input type="checkbox"/> EXECUTE
<input type="checkbox"/> FILE
<input checked="" type="checkbox"/> GRANT OPTION
<input checked="" type="checkbox"/> INDEX
<input checked="" type="checkbox"/> INSERT
<input checked="" type="checkbox"/> LOCK TABLES
<input type="checkbox"/> PROCESS
<input type="checkbox"/> REFERENCES
<input type="checkbox"/> RELOAD
<input type="checkbox"/> REPLICATION CLIENT
<input type="checkbox"/> REPLICATION SLAVE
<input checked="" type="checkbox"/> SELECT
<input checked="" type="checkbox"/> SHOW DATABASES
<input checked="" type="checkbox"/> SHOW VIEW
<input type="checkbox"/> SHUTDOWN
<input type="checkbox"/> SUPER
<input checked="" type="checkbox"/> TRIGGER
<input checked="" type="checkbox"/> UPDATE

Figure 66 Privileges of a generic user


Users and Privileges

User Accounts

User	From Host
(1) <anonymous>	%
<anonymous>	localhost
Admin	%
Developer	%
Developer	%
Quest	localhost
User	1
admin	%
customer	%
general	%
pma	localhost
root	127.0.0.1
root	localhost

Add Account
Delete
Refresh

Details for account User@1

Login
Account Limits
Administrative Roles
Schema Privileges

Schema	Privileges
dateng_database	INSERT, SELECT, TRIGGER
mydb	INSERT, SELECT, TRIGGER

Revoke All Privileges
Delete Entry
Add Entry...

Object Rights

☐ SELECT
☐ INSERT
☐ UPDATE
☐ DELETE
☐ EXECUTE
☐ SHOW VIEW

DDL Rights

☐ CREATE
☐ ALTER
☐ REFERENCES
☐ INDEX
☐ CREATE VIEW
☐ CREATE ROUTINE
☐ ALTER ROUTINE
☐ EVENT
☐ DROP
☐ TRIGGER

Other Rights

☐ GRANT OPTION
☐ CREATE TEMPORARY TABLES
☐ LOCK TABLES

Unselect All
Select "ALL"

Revert
Apply

Figure 67 Schema privileges of a generic user

The generic user was successfully created with global and schema privileges.

Exporting Data

The base will be exported using Workbench. The database will be exported with the following settings.

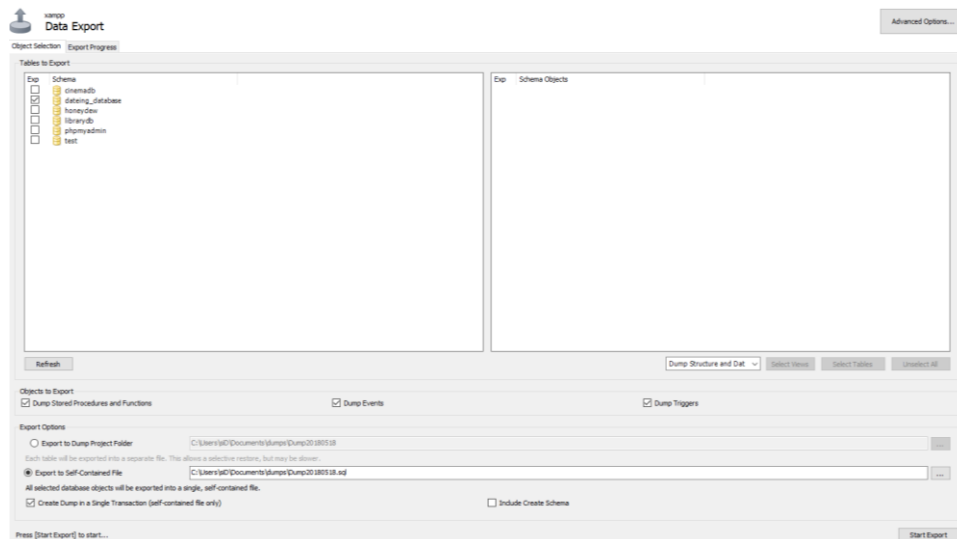


Figure 68 Exporting database

The database will now be exported to a SQL script file.

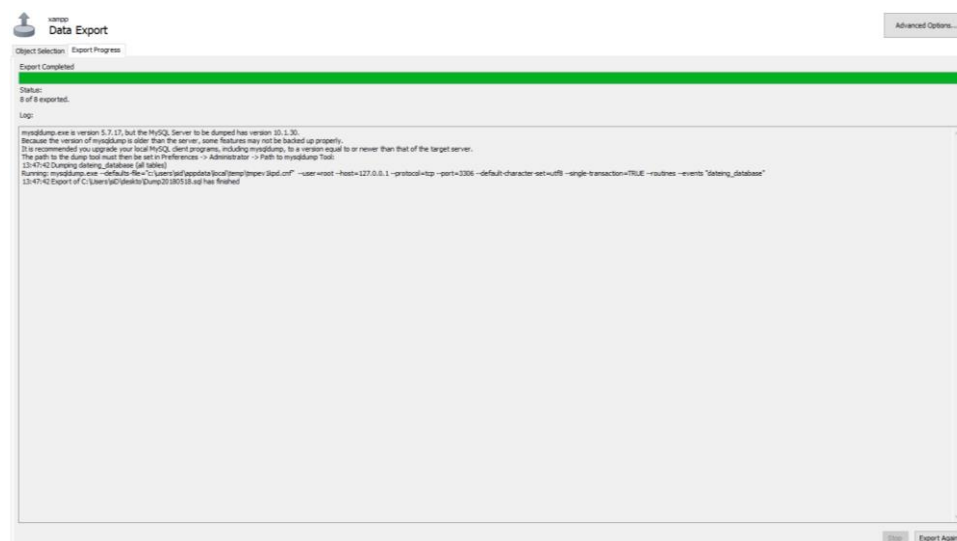


Figure 69 Successfully exported database

As seen in the figure above, the database was successfully exported.

Conclusion

As definitively shown in the sections above, a fully functional database has been designed and created.

All queries have been executed correctly. The database has assigned roles and privileges.

Each panel server a different purpose. From each panel the users are able to interact with the database in different ways. The database is inserted to and read from as well as updated and depleted. Only admin users in the admin frame can delete users.

Ultimately, the system is a success.

Question 2

2.1

Bophelo Hospital needs to beware of threats faced by modern database systems. Bophelo Hospital should implement comprehensive database security policies and procedures. Database security refers to the ongoing processing of shielding a database from both internal and external threats (Beyon-Davies, 2008; Connolly & Begg, 2015).

Within the context of the use of computer databases, a threat is a program or a person that places assets at a substantial risk or may cause the loss of assets (Whitman, et al., 2012; BSI Group, 2008). Regarding computer databases, a threat refers to situations or events that negatively affect a database system and thus the entities that interact with it. According to Connolly and Begg (2015), threats can be both intentional and unintentional.

Common threats that a database system faces include:

1. Database administration threats;
2. Database and data threats;
3. Hardware failure;
4. Users threats;
5. Programmer/operator threats;
6. Connections threats;
7. DBMS and other software threats; and;
8. Security threats.

Due to the vital roles that data plays in modern organisations the primary concern of a database administrator is to ensure the security of the database (Beyon-Davies, 2008). It is ultimately the responsibility of database administrators to ensure the database remains not only operational but secure (Whitman, et al., 2012). This is indicated in the figure that follows.

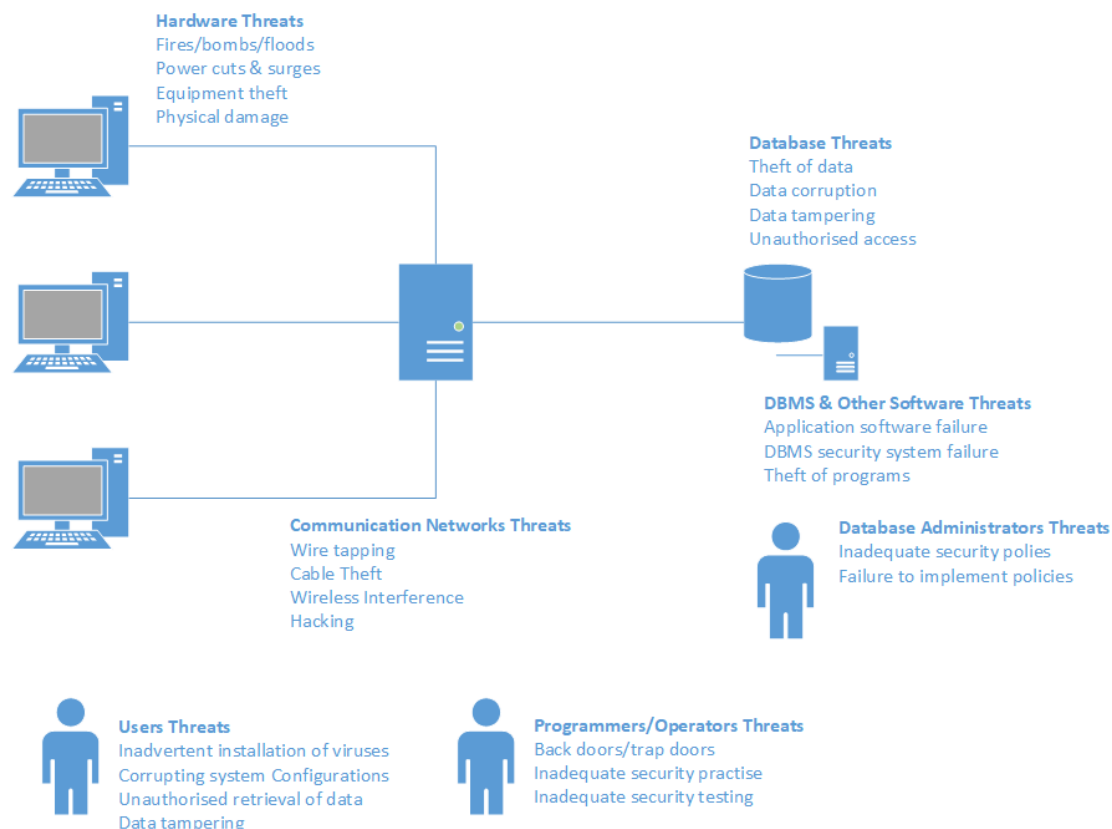


Figure 70 Database threats

Within the context of the use of computer databases, database threats and data threats refer to threats that may cause data loss or data corruption as well as the unauthorised access to data and data theft (Connolly & Begg, 2015).

All computer system can potentially face hardware failure. Some common causes of hardware failure include the age of equipment, theft of equipment and power surges as well as floods, fire, tsunamis or other natural disasters (Connolly & Begg, 2015).

Within the context of the use of computer databases, user threats are caused by a user of a system causes a system to fail or experience a security breach. Common examples of user threats include the inadvertent installation of viruses, modifying or corrupting system configurations and the unauthorised retrieval or tampering of data (Connolly & Begg, 2015). According to Mackey (2008), humans are the greatest threat to an IT-based system. The education of end-users is one the most effective ways to combat user threats (Whitman, et al., 2012).

Within the context of the use of computer databases, programmer threats caused by the programmers who create a database system. These threats include unresolved bugs, the

creating of trap doors and the inadequate implementation of security practices. It is crucial to eliminate these threats during the testing phase of the system's development lifecycle.

Within the context of the use of computer databases, connection threats refer to any potential threats caused by a system being connected to a computer network, such as the Internet. Common network threats include wire-tapping, wireless interception and hacking as well as cable theft and wireless interference (Connolly & Begg, 2015). According to Whitman, et al. (2012) the risk of threats increases when a system is connected to the Internet.

Database Management System (DBMS) and software threats refer to the failure of a database DBMS to operate as expected or the failure of the application software (Connolly & Begg, 2015). Examples of these threats include the failure of important application software, such as antiviruses, and the failure of DBMS security measures to trigger.

Cybersecurity involves securing all resources connected to the internet (Whitman, et al., 2012). According to Castano, et al. (1995) and Connolly & Begg (2015), possible data security threats include:

1. Data Theft;
2. The loss of data confidentiality;
3. The loss of data integrity;
4. The loss of data availability;
5. Fraud; and;
6. The release of confidential data.

All the above listed threat can be mitigated through effective threat counter measures and risk management.

Database Concurrency control and recovery are vital in protecting a database against becoming inconsistent (Connolly & Begg, 2015). Concurrency control and recovery are essential to the operation of any modern database system (Connolly & Begg, 2015). Concurrency control ensures that multiple users can access a database simultaneously (Connolly & Begg, 2015). In conclusion, it is vital that Bophelo Hospital implements both recovery and concurrency control mechanisms. The effective implementation of effective these mechanisms will ultimately ensure the longevity and thus the productivity of Bophelo Hospital.

2.2

There are several potential countermeasures that Bophelo Hospital can employ to combat the threats previously identified. As stated by Beyon-Davies (2008), database security is the ongoing processes of defending a database from threats. Due to the vital roles that data plays in modern organisations the primary concern of a database administrator is to ensure the security of the database (Beyon-Davies, 2008).

Multiple levels of security can be implemented to help protect Bophelo Hospital against threats to their sensitive information. Whitman, et al. (2012) suggests the implementation of layers, or spheres, of security.

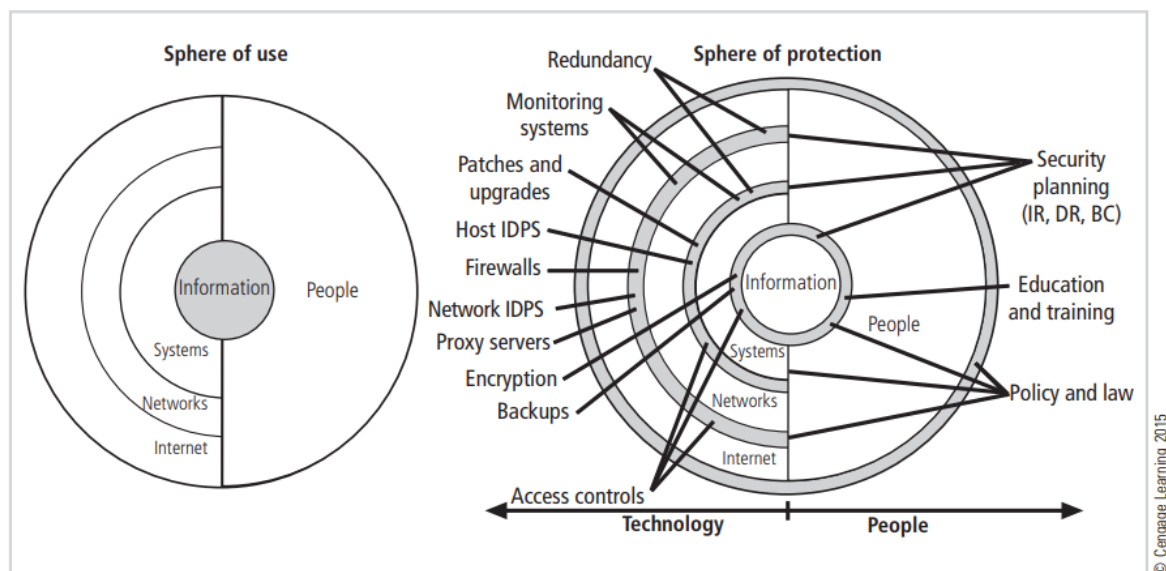


Figure 71 Layer of Security (Whitman, et al., 2012)

As can be shown in the figure above, various levels of information security should exist, to ensure the protection of Bophelo Hospital's information (Whitman, et al., 2012).

According to Castano, et al. (1995) database administrators need to implement both computer-based and non-computer-based countermeasures to effectively protect a database from threats. Common computer-based countermeasures:

1. Suitable authentication and authorisation;
2. Implementing views;
3. Encrypting sensitive information;
4. Ensuring data integrity;
5. Establishing backups and recovery procedures; and;

6. Implementing Redundant Array of Independent/Inexpensive Disks (RAID) (Beyon-Davies, 2008).

Database authorisation refers the practice of granting privileges to specific users that allows them to access and interact with d database system (Beyon-Davies, 2008). The effective implementation of authorisation will ensure that only legitimate users have access to parts of a database that they need (Beyon-Davies, 2008). Authorisation ultimately leads to a more secure database. (Beyon-Davies, 2008) defined authentication as mechanisms that determine if a certain is who they say they are. The figure below is a graphical representation of the authentication authorisation process.

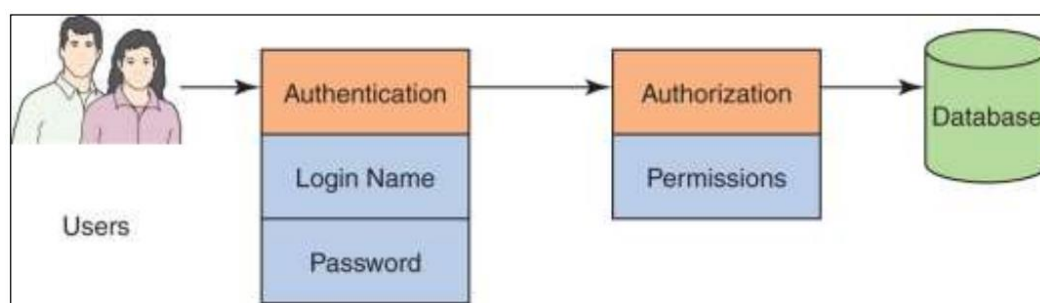


Figure 72 Database authentication and authorization (Kroenke & Auer, 2013)

As seen above, only authenticated users are allowed to access the sections of the database that they are authorised to use. Suitable authentication and authorisation methods can be used to combat the threat of ransomware and other viruses.

A view is a virtual representation of a table of a table stored in a database (Groff & Weinberg, 2002). As views are defined by queries they may be constructed from data taken from multiple tables (Beyon-Davies, 2008). The contents of a view are defined by a query. Views create the illusion of an additional table existing within the database, therefore, the information contained within a view cannot be altered (Beyon-Davies, 2008). This is used to improve the security of databases (Connolly & Begg, 2015).

Encryption involves hiding of sensitive information so that it is in a form that is not readable unless decrypted. This ensures that if valuable information is stolen, the thief will be able to understand and use the data without the key. Encrypting the data sent networks, such as the internet, will increase the overall security and integrity of the network. The encryption of data will help Bophelo Hospital.

Database integrity is the ongoing process of ensuring that all data stored in a database is accurate (Connolly & Begg, 2015). A key component of database integrity ensures that all of the data inserted into a database is in the correct format (Connolly & Begg, 2015). Database integrity is vital to any database system to guarantee that no incorrect results are returned (Connolly & Begg, 2015).

Beyon-Davies (2008) suggest the establishing recovery procedures and periodically making backups of log files and the database itself. These backups should be stored in a separate location to the physical hardware of the database (Connolly & Begg, 2015). The effective creation of backup will help to protect Bophelo Hospital against the loss of important data.

Redundant Array of Independent/Inexpensive Disks (RAID) creates a “reconstruction map” so that lost data may be recovered if a hard drive fails (Stair & Reynolds, 2016). RAID minimise the impact of a hard drive failure by dispersing the data across various hard drives (Whitman, et al., 2012). These features allow the RAID to be an effective countermeasure against hardware failure. The effective implementation of the RAID will help to protect Bophelo Hospital against the loss of important data.

In addition to computer-based countermeasures Beyon-Davies (2008) suggest the implementation of non-computer-based countermeasures, including:

1. Establish and adhere to security policies;
2. Separate person duties;
3. Store computer hardware behind secure access points; and;
4. Store backups and hard copies off-site, fireproof, storage (Beyon-Davies, 2008; Connolly & Begg, 2015).

It is vital that database administrators communicate with other members of their organisation to ensure that these policies are maintained and upheld to an appropriate standard (Beyon-Davies, 2008).

There are various countermeasures that Bophelo Hospital can employ to help protect their system against threats. Bophelo Hospital should implement both computer-based, and non-computer-based countermeasures. The correct implementation of effective countermeasures will ultimately ensure the longevity and thus the productivity of Bophelo Hospital.

2.3

It is vital that Bophelo Hospital is able to function effectively on a large scale. A major concern of modern database is ensuring that the multiple users can access database resources simultaneously (Connolly & Begg, 2015). As previously stated, Database integrity is the ongoing process of ensuring that all data stored in a database is accurate (Connolly & Begg, 2015). Effective concurrency control is vital in ensure database integrity.

Concurrency control is the process of handling simultaneous database operations without them conflicting with each other (Connolly & Begg, 2015). Although two, or more, operations may be correct on their own, the interleaving of database operations may result in an incorrect result being produced (Connolly & Begg, 2015; Barghouti & Kaiser, 1991). This may result in the database entering an inconsistent state. Remaining isolated is an important property of any database transactions (Bernstein & Newcomer, 2009). This transaction isolation is vital in ensuring that a database remains consistent (Connolly & Begg, 2015).

According to Barghouti and Kaiser (1991) and Connolly and Begg (2015), inadequate concurrency control may lead to the following errors:

1. Uncommitted dependency;
2. Inconsistent analysis; and;
3. Lost update.

A lost update occurs when one apparently successful transaction is overridden by another apparently successful transaction (Chhanda, 2008). This occurs when a transaction reads an intermediary result before it is committed to the database (Connolly & Begg, 2015). This will cause the database to enter an inconsistent state. The figure below is a representation of the lost update error provided by Connolly and Begg (2015).

Time	T ₁	T ₂	bal _x
t ₁		begin_transaction	100
t ₂	begin_transaction	read(bal _x)	100
t ₃	read(bal _x)	bal _x = bal _x + 100	100
t ₄	bal _x = bal _x - 10	write(bal _x)	200
t ₅	write(bal _x)	commit	90
t ₆	commit		90

Figure 73 Lost update example (Connolly & Begg, 2015)

As seen above, a lost update problem has occurred causing the transaction to execute and resulting in the database entering an inconsistent state.

According to (Connolly & Begg, 2015), an inconsistent analysis problem occurs when a transaction requires various values and one of which is altered during the transition. This is also known as a dirty read or an unrepeatable read (Connolly & Begg, 2015). An unrepeatable cannot be repeated. This will cause the database to enter an inconsistent state. The figure below is a representation of the lost update error provided by Connolly and Begg (2015).

Time	T ₅	T ₆	bal _x	bal _y	bal _z	sum
t ₁		begin_transaction	100	50	25	
t ₂	begin_transaction	sum = 0	100	50	25	0
t ₃	read(bal _x)	read(bal _x)	100	50	25	0
t ₄	bal _x = bal _x - 10	sum = sum + bal _x	100	50	25	100
t ₅	write(bal _x)	read(bal _y)	90	50	25	100
t ₆	read(bal _z)	sum = sum + bal _y	90	50	25	150
t ₇	bal _z = bal _z + 10		90	50	25	150
t ₈	write(bal _z)		90	50	35	150
t ₉	commit	read(bal _z)	90	50	35	150
t ₁₀		sum = sum + bal _z	90	50	35	185
t ₁₁		commit	90	50	35	185

Figure 74 Inconsistent analysis problem (Connolly & Begg, 2015)

As seen above, an inconsistent analysis problem has occurred causing the transaction to execute and resulting in the database entering an inconsistent state. From here, the database will need to be restored. This problem could have been avoided through the effective implementation of a correct database concurrency control mechanism.

Concurrent access is relatively easy to implement if all users are only reading data, as there is no way that they can interfere with one another (Connolly & Begg, 2015). Database Management Systems (DBMSs) must be able to facilitate multiple users accessing database resources simultaneous without leaving the database in an inconsistent state (Beyon-Davies, 2008). One of the advantages of using a DBMS is that they inherently allow for multiple users to make use of database resources concurrently (Beyon-Davies, 2008).

According to Connolly and Begg (2015), there are two primary types of concurrency control, namely:

1. Locking;
2. Timestamping;
3. Cascading rollback;
4. Concurrency control with index structures; and;
5. Logging (Connolly & Begg, 2015).

Connolly and Begg (2015) stated both concurrency control mechanisms listed above are pessimistic approaches. Pessimistic approaches pre-emptively prevent conflicts from occurring (Connolly & Begg, 2015). Both have their own advantages and disadvantages.

Locking denies the access of other transactions to guarantees that transactions are not tampered with (Connolly & Begg, 2015). According to Connolly and Begg (2015), locking is the most commonly implemented form of database concurrency control. Two-phase locking is a variation of database locking (Connolly & Begg, 2015). Locking is a potential concurrency control mechanism that Bophelo Hospital could make use of.

Locking is not always possible. According to Connolly and Begg (2015), issues may occur when releasing locked data. The cascading rollback concurrency control mechanism circumvents these issues (Connolly & Begg, 2015). The figure below is a graphical example of cascading rollback. It has been provided by Connolly and Begg (2015.)

Time	T ₁₄	T ₁₅	T ₁₆
t ₁	begin_transaction		
t ₂	write_lock(bal_x)		
t ₃	read(bal_x)		
t ₄	read_lock(bal_y)		
t ₅	read(bal_y)		
t ₆	bal_x = bal_y + bal_x		
t ₇	write(bal_x)		
t ₈	unlock(bal_x)	begin_transaction	
t ₉	:	write_lock(bal_x)	
t ₁₀	:	read(bal_x)	
t ₁₁	:	bal_x = bal_x + 100	
t ₁₂	:	write(bal_x)	
t ₁₃	:	unlock(bal_x)	
t ₁₄	:	:	
t ₁₅	rollback	:	
t ₁₆		:	begin_transaction
t ₁₇		:	read_lock(bal_x)
t ₁₈		rollback	:
t ₁₉			rollback

Figure 75 Cascading rollback (Connolly & Begg, 2015)

As seen above, locks are only released at the end of each transition. This is done to ensure that each transition remains atomic, consistent, isolated and durable. Cascading rollback is a potential concurrency control mechanism that Bophelo Hospital could make use of.

According to Connolly and Begg (2015), concurrency control with index structures involves treating each page as an index applying Two-phase locking where needed. This is done to guarantee that each transition remains atomic, consistent, isolated and durable. Connolly and Begg (2015) note that this control mechanism will lead to a large section of the database being frequently locked as indexes is often accessed. For this reason, it is not recommended for Bophelo Hospital to make use of concurrency control with index structures.

Timestamping is a form of concurrency control that Bophelo Hospital may wish to implement. Timestamping stamps each transaction with a timestamp and given older transition a higher priority than newer transactions (Connolly & Begg, 2015). This is done to ensure that each transition remains atomic, consistent, isolated and durable. It is important to note that all timestamps need to be unique (Connolly & Begg, 2015). Timestamping is a potential concurrency control mechanism that Bophelo Hospital could make use of.

Multi-version timestamp ordering is one form of timestamping that the hospital may wish to make use of. Ultimately, the multi-version timestamp is a better version of timestamping. Multi-version timestamp allows the transaction to read various versions of stored data (Connolly & Begg, 2015). This can increase the performance of the control mechanism and thus increase the performance of the database system as a whole.

Bernstein and Newcomer (2009) noted that although the concept of logging is relatively simple, logging can have an unforeseen impact on both performance and data correctness. This is unforeseen complexity can be attributed to the delays that take place before a transaction is committed (Bernstein & Newcomer, 2009). According to Elmasri and Natathe (2011), locking is used in most commercial DBMSs. This makes it ideal for the hospital.

It is vital to maintaining database integrity (Connolly & Begg, 2015). In conclusion, Bophelo Hospital needs to effectively implement modern concurrency control mechanisms. This done so that multiple users can access a database simultaneously, or concurrently. Furthermore, concurrency control mechanisms will help ensure that transactions remain isolated so that the database remains consistent (Connolly & Begg, 2015).

2.4

In order to effectively provide healthcare services, Bophelo Hospital needs to establish database recovery methods. Database recovering is the processes of reverting a database back to a stable state after a failure (Connolly & Begg, 2015). Database failures may result in the unintentional, or intention, data corruption, data loss or denial of services (Connolly & Begg, 2015).

It is the responsibility of the DBMS to ensure that the database can be restored to a consistent state after a failure has accorded (Beyon-Davies, 2008). This should be done by the network manager/database manager of the Hospital. DBMS need to be able to recover from both hardware and software failure (Beyon-Davies, 2008).

Database failure can be attributed to numerous sources each of which needs to be dealt with in a distinct manner (Connolly & Begg, 2015). Some common causes of hardware failure include the age of equipment, theft of equipment and power surges as well as floods, fire, tsunamis or other natural disasters (Connolly & Begg, 2015). There are two primary types of a database failure, failures that affect main memory and failures that effect non-volatile (secondary) storage (Connolly & Begg, 2015). As stated by Beyon-Davies (2008), DBMS need to be able to recover from both hardware and software failure. All modern DBMS provide the following the following recover mechanisms:

1. Backup mechanism;
2. Logfiles; and;
3. Mechanism to restores the database (Connolly & Begg, 2015).

The table below outlines some common database failures and describes them.

Table 14 Common causes of database failure

Failure	Description
System crashes	According to Connolly and Begg (2015) system crashes can be caused by hardware and software errors. System crashes can result in the loss or corruption of main memory (Connolly & Begg, 2015).
Media failure	Media failure can result in losing parts of secondary storage (Connolly & Begg, 2015). Common media failures include unreadable media and head crashes (Connolly & Begg, 2015).
Application software errors	Application software errors, such as bugs and logical errors, with programs that are accessing a database, may cause transactions to fail (Connolly & Begg, 2015).

Failure	Description
Natural disasters	Natural disasters, such as tsunamis, fires, and floods, can lead to equipment failure and breakages (Connolly & Begg, 2015).
Human error	Human error, careless operation and the unintentional damages can lead to equipment failure and loss of critical business information (Connolly & Begg, 2015).
Sabotage	Sabotage, terrorism or the intentional destruction of hardware facilities will cause equipment failure and thus downtime (Connolly & Begg, 2015).

According to Connolly and Begg (2015), database failure can affect a database system in one to two ways, namely: the loss of main memory or the loss of the disk copy of the database. Main memory refers to the physical disk and storage locations that the data is stored on (Connolly & Begg, 2015).

According to Connolly and Begg (2015), the recovery procedure that should be used depends on the type of failure and extent of the damage it caused. Failures and the damage they caused can be classed into two primary categories, namely:

1. The database and its data stored has been irreversibly destroyed; and;
2. The database has not been physically damaged or destroyed by the data has become inconsistent (Connolly & Begg, 2015).

There are three primary techniques for recovering inconsistent data, deferred update; immediate update and shadow paging (Connolly & Begg, 2015). These are all forms of automated recovery.

The deferred update recovery protocol does not update the database until the transaction reaches its commit point (Connolly & Begg, 2015). Thus, if a transaction fails before the commit point has been reaching no changes would have been made to the database and therefore no changes need to be undone (Connolly & Begg, 2015). The database will remain in a consistency state. Connolly and Begg (2015) noted after a failure has occurred that it may be needed to redo all provisions updates on committed transactions as they may not have been committed.

The immediate update recovery protocol updates the database as transactions occur regardless if they have reached the commit point (Connolly & Begg, 2015). Connolly and Begg (2015) stated that after a failure has occurred noted that it may be needed to redo all previous updates on committed transactions as they may not have been committed.

Both the deferred update protocol and the immediate update protocol allow a corrupted database to be restored to a consistent state. Both protocols use log files to protect against system failures (Connolly & Begg, 2015).

Lorie (1977) suggested shadow paging as an alternative to log-based recovery protocols. Shadow paging maintains two-page tables for a transaction, a current page table and shadow page table (Connolly & Begg, 2015). The transaction only interacts with the current page table (Connolly & Begg, 2015). As the shadow page table is not altered by the transaction it can be used as a restore point if a system failure occurs (Lorie, 1977). Once the transaction has been completed successfully the shadow page table is replaced by the updated current page table thus updating the database (Connolly & Begg, 2015). According to Connolly and Begg (2015), shadow paging has less overhead compared to the immediate update and deferred update recovery protocols as there is no need to maintain a log file.

Database recovery is key to any database system. It is vital that the DBMS ensures that the database remains not only reliable and consistent, but that entity integrity is maintained (Connolly & Begg, 2015).

There are various recovery mechanisms that Bophelo Hospital can employ to help protect their system against database failure and other data threats. The hospital should implement some form of automated recovery. The correct implementation of effective and efficient and recover mechanisms will ultimately ensure the longevity and thus the productivity of Bophelo Hospital.

Bibliography

- Barghouti, N. S. & Kaiser, G., 1991. *Concurrency control in advanced database*. New York: ACM Computing Survey.
- Bennet, S., McRobb, S. & Farmer, R., 2010. *Object-Oriented Systems Analysis and Design Using UML*. 4th ed. London: McGraw Hill.
- Bernstein, P. A. & Newcomer, E., 2009. *Principles of Transaction Processing*. 2nd ed. Burlington: Morgan Kaufmann.
- Beyon-Davies, P., 2008. *Database Design*. 3rd ed. Basingstoke: Palgrave Macmillan.
- BSI Group, 2008. *BS ISO/IEC 27005:2008*. 1st ed. London: BSI Group.
- Castano, S., Fugini, M., Martella, G. & Samarati, P., 1995. *Database Security*. Boston: Addison-Wesley.
- Chhanda, R., 2008. *Distributed Database Systems*. London: Dorling Kinderley.
- Connolly, T. M. & Begg, C. E., 2015. *Database Solutions: A step-by-step approach to building databases*. 6th ed. Harlow: Pearson Education Limited.
- Deitel, P. & Deitel, H., 2012. *Java : How to Program*. 9th ed. Boston: Prentice Hall.
- Drake, P., 2014. *Data Structures and Algorithms in Java*. 1st ed. Harlow: Pearson Education.
- Elmasri, R. & Natathe, S. B., 2011. *Database Systems: Model, Languages, Design and Application Programming*. Boston: Pearson.
- Gaddis, T., 2016. *Programming Logic & Design*. 4th ed. Boston: Pearson.
- Groff, J. R. & Weinberg, P. N., 2002. *SQL: The Complete Reference*. 2nd ed. New York City: McGraw-Hill.
- Kroenke, D. M. & Auer, D. J., 2013. *Database concepts*. 6th ed. New Jersey: Prentice Hall.
- Lorie, R. A., 1977. Physical Integrity in a Large Segmented Database. *ACM Transactions on Graphics*, 2(1), pp. 91-104.
- Mackey, D., 2008. *Web Security for Network and System Administrators*. Stamford: Thomson Course Technology.
- Nielsen, J., 1994. Heuristic evaluation. In: *Usability Inspection Methods*. New York: John Wiley & Sons, pp. 25-62.
- Preece, J., Rogers, Y. & Sharp, H., 2015. *Interaction Design: Beyond Human-Computer Interaction*. New Jersey: John Wiley & Sons.

Pretorius, C. M. & Erasmus, G. H., 2012. *Basic Programming Principles*. 2nd ed. Cape Town: Pearson Education South Africa.

Rogers, Y., Sharp, H. & Preece, J., 2011. *Interaction Design: Beyond human-computer interactions*. 3rd ed. Chichester: Wiley.

Stair, R. M. & Reynolds, G. W., 2016. *Principles of Information Systems*. Boston: Cengage Learning.

Valacich, J. S., George, J. F. & Hoffer, J. A., 2015. *Essentials of Systems Analysis and Design*. Global Edition ed. New Jersey: Prentice Hall.

Weisfeld, M., 2013. *The Object-Oriented Thought Process*. 4th ed. Boston: Addison-Wesley.

Whitman, M., Coles, M. & Mattord, H., 2012. *Principles of information security*. 4th ed. London: Cengage Learning.

