Entity Relationship Diagram Mapping

by

Cheryl Dunn

Comp 4905

Prof. Lou D. Nel

School of Computer Science

Carleton University

Ottawa, Ontario

December 15, 2017

Abstract

Entity Relationship Mapper (ER Mapper) is an android app that takes an Entity-Relationship diagram (ER Diagram) and maps it to its Relations, finds its Functional Dependencies, perform normalization and creates a relational database. Users can draw an ER Diagram on a canvas, such that they can create, remove and edit Entity, Attribute and Relationship objects from the canvas. When the user is satisfied with their diagram, they can select to save or normalize the diagram into a relational schema in third normal form maintaining lossless join and dependency preservation properties to create a database.

Acknowledgments

I would like to thank Professor Louis Nel for his support and expert advice as well as for use of his JavaFXNormalizer code which allowed for Functional Dependencies to be normalized preserving lossless join and dependency preservation properties.

Table of Contents

List of Figures

List of Tables

1. Introduction

Computers have countless applications in the world ranging from text editors to games to security and much more. An important part of each of these application is storing information. In some scenarios, the best way to do so is by using a relational database. Relational databases organize a collection of data as schemas in tables. When designing a database, it is important to organize the data so that it is modeled in a realistic way to support the real world that it is modelling as well as reduced redundancy and errors. An Entity Relationship (ER) Diagram is a widely used method for conceptualizing and visualizing the logical structure of a Relational database. There is also an Extended ER Diagram that hands specializations and generalizations, which will not be used in the completion of this project. By creating an ER Diagram and normalizing it to its functional dependencies database designers can easily and realistically model the real world as a database. The purpose of this project is there for to create an android app (ER Mapper) that allows users to draw an ER diagram and it automatically normalized in order to generate a database. The ER Mapper app has a practical use as in a lecture, or meeting environment this app can help teach students or clients about ER Diagrams and the normalization process. ER Mapper provides an interface that allows for users to interact with the diagram and get an idea of the ER Diagrams design process along with how the normalization process occurs. As an android app ER Mapper can be used on any android device, though a tablet or device with a better bigger screen is best. This allows for users to get an idea quickly and efficiently of what their database design. This report Demonstrates how to setup and run the ER mapper program and explains how the program talks an ER Diagram, maps it to its Relations and Functional Dependencies to create a database.

2. Setup

Clone the Git repository from https://github.com/CD11/ERmapper or unzip the project from its Compressed Zip file, into the repository of your choice.

2.1 Downloading Android studio

1. Install Android studio 3.0 with
   - JRE: 1.8.0_152-release-9159b01
   - JVMOpenJDK 64-bit Server VM by JetBrains s.r.o

2. In Android studio go to File -> open an existing android studio project
   - A dialog will pop up, select the directory where you saved ERMapper, and select **ERMapper->ERMapper->APP**.

2.2 Using emulator in android studio

1. In Android studio go to **Tools** > **Android** > **SDK Manager**
2. When the pop up screen opens select **Nougat 7.0.0 or Nougat 7.1.1**
   - (The program requires a min API Level of 24)
3. In Android studio go to **Tools** > **Android** > **AVD Manager**
4. When the pop up screen opens select create virtual device
   - Go to **Tablets** > **Nexus 10** then press Next
5. On the Next screen make sure the SDK that you choose from step 2 is selected, and then click next and finish

2.3 Using an android device.

1. Install OEM USB drivers corresponding to the device you are using, go to https://developer.android.com/studio/run/oem-usb.html for more information.
2. Enable USB Debugging on your android device
   - Open the **Settings** app.
   - (Only on Android 8.0 or higher) Select **System**.
   - Scroll to the bottom and select **About phone**.
   - Scroll to the bottom and tap **Model number 7** times.
   - Return to the previous screen to find **Developer options** near the bottom.

3. Running the software

Once the code has been imported and all the necessary software is set up you can run the program. To run the program, press the green arrow. This will launch a prompt that asks the user to select a device, emulated or physical, and press OK. This will install the software onto the android device or launch the emulator. It may take a minute or two to download and start up the system, but when it is ready the ER Mapper will launch.

3.1 Create a New Drawing

From the initial screen select new ER Diagram, this will launch a new activity with a blank canvas, and some button along the bottom.

3.2 Creating Objects

Press either the Entity or Attribute button to create either object. They will appear in the Top left side of the canvas; an Entity will be square, and an Attribute will be oval. Both can be moved around by clicking and dragging them around the screen.

Press the Relationship object to create a relationship. Once the button is selected, click on either an entity or attribute object, a line will appear that connects the object to the user's mouse or finger. Click the second object to create the connection. If for some reason the relationship is not created on the first try or the line is no longer available, click on the relationship button again. When a relationship between two Entitles is created, a "1" will appear above the line next to the object, this is denoting relationship cardinality, and can be set to 1, N or M.

Note:

1. Double clicking an Entity can set it to weak,
2. Every Foreign key must have an identifying primary key with the same name
3. Double clicking and Attribute can set it to primary
4. All objects have a name that can be edited by clicking it, however it uses an onFocusChange listener, so after you edit the name you must click something else on the screen, or press tab or enter.

3.3 Saving the Diagram

By pressing save the diagram you will save from the drawing screen, the diagram in XML format to the android device.

3.4 Normalize the Diagram

Pressing Normalize from the drawing screen will launch a new Activity that automatically converts all entity objects to relations, creating a relational Schema, finding all functional dependencies and normalizing the schema into third normal form. From here one will be able to see all attributes, functional dependencies, min cover, candidate keys, and lossless join /dependency preserving tables.

3.5 Create a Database

By pressing the create database button the system will open a connection to a SQLite database and create a new database inserting each relation from the schema as a table in the database.

3.6 Accessing Files

Once an ER Diagram has been saved, or a diagram has been created they are saved to the internal storage of the android device. Within android studio you can access the files from your emulated or physical device by

1. **view->tool window ->device file explorer,** this will cause a file explorer to appear on the right-hand side of android studio
2. In the prompt go to data->data->cd.com.ermapper
3. Database files are stored in the **database** folder, and xml files are stored in the **files** folder.

4. Research

Within this section key ideas and concepts regarding ER Diagrams, FDNormalization and Databases are identified and discussed. All research comes from *the Book Fundamentals of Database* Systems by Ramex Elmasri & Shamkant B. Navathe, as well as from COMP3005 Winter 2015 course notes provided by Professor Louis Nel.

4.1 ER Diagram Components

To draw an ER Diagram, it is important to understand what they are, along with its components. Components include symbols that are used to represent a different concept in the diagram including Entities, Attributes and Relationships. It should be noted now that the ER Mapper app will only work with relational databases which is a database that uses tables and specific constraints to model data.

4.1.1 Entity Relationship Diagrams

ER Diagrams are a visualization that represents relationships between Entities. Entities, modeled as squares, represent a real-world concept, and create the tables in your Relational database. Attributes, modelled as ovals, represent properties of their corresponding Entity or Relationship and are stored as columns of the Relational table in a database. Every Entity can have multiple Attributes; however, it requires at least one Attribute to be a primary key (which uniquely identifies each row in the table). In an ER Diagram the primary key attribute(s) are

identified with an underline underneath their name. A Relationship is a line that connects Attributes and Entities, they may also indicate the cardinality of the Relationship.

Figure 1. ER Diagram Symbol, from the book *Fundamentals of Database Systems* below shows the Symbols that may appear in an ER Diagram.

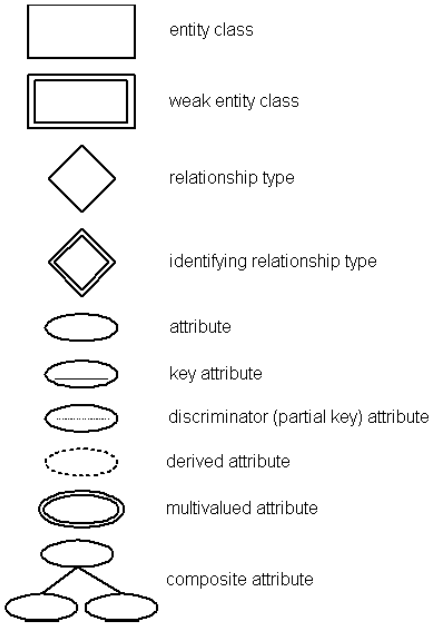

Figure 1. ER Diagram Symbol

4.1.2 Entity objects

An Entity object is a distinguishable object that is part of the mini-world modeled by the database. In a Relational database, an Entity will be a Relation/Table. Entities can have different types. Regular Entities have a primary key Attribute, and are Strong Entities. Weak Entities cannot be distinguished by themselves as they do not have a unique key Attribute. For a weak Entity to be identified a primary key is defined using a foreign key to its strong Entity and always has a total participation constraint. *Figure 2. Weak Entity Example* shows a weak Relationship where Grade is the weak Entity, and class is the Strong Entity. The primary key of grade is "C#", "S#", where code references the primary key of "S#" of Student and "C#" of Course.

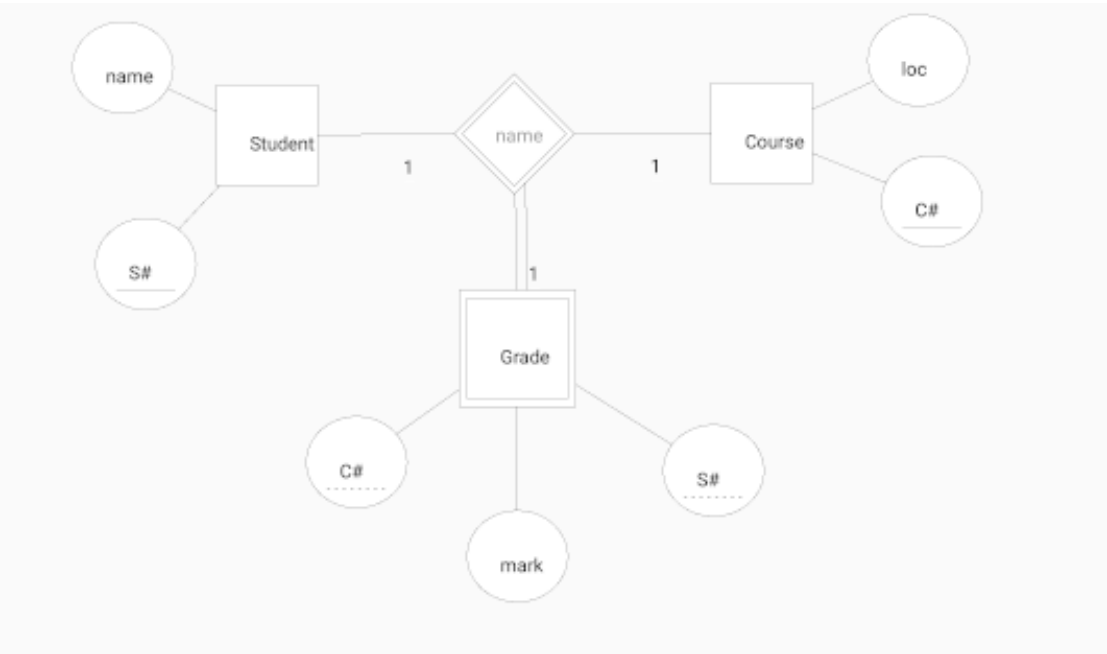

Figure 2. Weak Entity Example

4.1.3 Attribute

An Attribute is a property describes an Entity object, and stored as a column in a relational table. An Attribute property may also describe a property of a Relationship. A candidate key is the set or primary keys that uniquely identify the table. A super key is a set of Attributes in a relation schema, where no two tuples in the Attribute set will have $t_1[S] = t_2[S]$. The removal of any Attribute from the super key will prevent it from being a super key. A Foreign Key Attribute is an Attribute that references a primary key Attribute of another table. Finally, a Composite Attribute is an Attribute that is composed of several components

4.1.4 Relationship

A Relationship relates two or more Entities with a specific meaning. The degree of Relationship is the number of Entities that participate in a Relationship. A binary relationship has a degree of 2, a ternary relationship has a degree of 3 and a n-ary relationship has a degree of N. Each participating Entity in a Relationship has a cardinality which can be 1:1, 1: N or M: N. *Figure 3. Relationship Types*, is an example that depicts different types of participation and cardinality of Relations. An Identifying Relationship type relates a weak Entity type to its owner.

In *Figure 2. Weak* Entity Example, the Relationship Depends on represents an identifying Relationship.

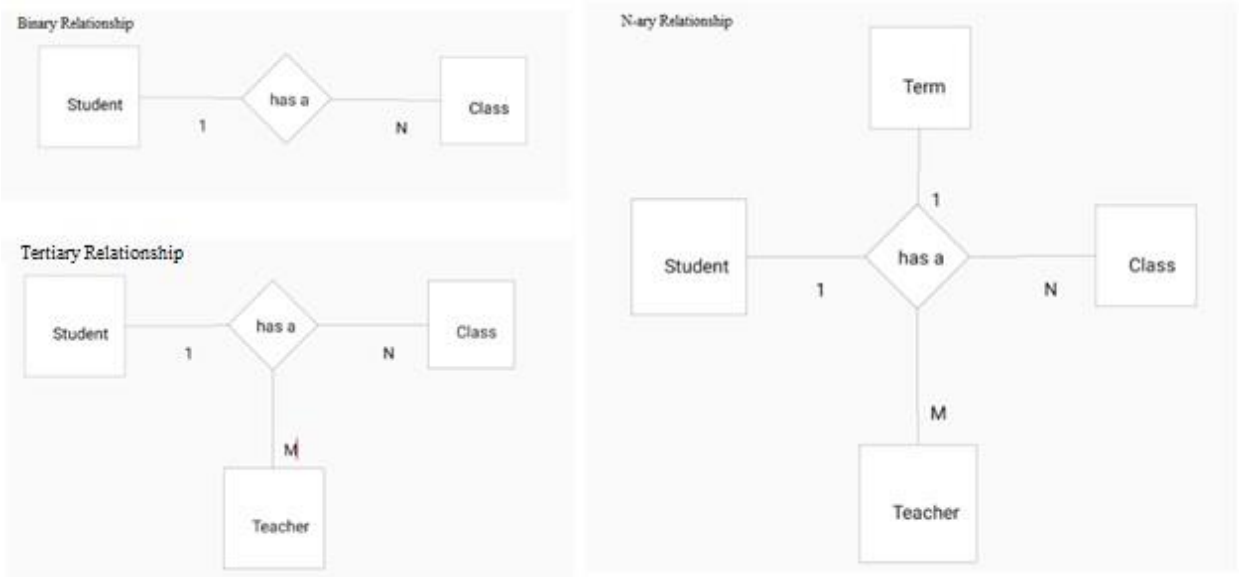

Figure 3. Relationship Types

4.2 Relation Schema Mapping and Normalization

The outcome of this project is a relational schema in third normal form that is mapped from the ER Diagram drawn by a user that can be mapped to a database. To properly create a relational schema an algorithm must be used to: convert n-ary Relationships to binary Relationships, find Functional Dependencies and perform normalization.

4.2.1 Decomposing Relationships

Before normalizing the diagram, The ER Diagram gets mapped to a RelationSchema, by looking at each Relationship and its Entities and deriving Relations. As discussed in the research section, a binary Relationship is a Relationship between two Entity objects, a ternary has three Entity objects and n-ary has n Entity objects. Relationships with a higher degree make it harder to specify the constraints of a Relation. Therefore; we can decompose the higher degree Relationships to binary to simplify the process. Decomposition to binary Relationships can be done following steps from *Fundamentals of Database Systems*:

1. For each ternary/ n-ary Relationship type R, where n>2, create a new Relationship S to represent R
2. Include as foreign key Attributes in S the primary keys of the Relations that represent the participating Entity types
3. Also include any simple Attributes of the n-ary Relationship type (or simple components of composite Attributes) as Attributes of S

4.2.2 Finding Functional Dependencies

Once the ER Diagram has been mapped to a relational schema, it is possible to identify the constraints for each Entity in the system. As defined in section 14.2.1 in *Fundamentals of Database systems:*

> "A Functional Dependency, denoted X -> Y, between two sets of Attributes X and Y that are subsets of [Relation] R specifies a *constraint* on the possible tuples that can form a Relation state r of R. The constraint is that, for any two tuples t1 and t2 in r have $t_1[x] = t_2[x]$, they must also have $t_1[Y] = t_2[y]$."

This definition means that for any value Y in a tuple is defined by X and it can be said that Y is functionally dependent on X. A Functional Dependency can have multiple different values, every Attribute of X is called the left had side, and every Attribute of Y is called the right-hand side. A set of Functional Dependencies exists for each Relation, where each Relation also must have a primary key. By identifying the Functional Dependencies, it is possible to find redundant and trivial information.

4.2.3 Performing Normalization

To ensure the relational schema has a good design without redundant or trivial information the normalization process is used. Normalization is the process of decomposing the relational schema into smaller relations using their candidate keys and functional dependencies (Nel, 2015c). Boyce Codd proposed the normalization process using First Normal Form, Second Normal Form and Third Normal Form. Boyce Codd later suggests Boyce-Codd Normal Form and Fourth Normal Form, but for this project, third normal form is sufficient. In order to be in Third Normal form, each Relation must also meet the requirements for First and Second normal forms which are as follows:

1. First Normal Form: has no composite/multivalued Attributes along with nested Relations
2. Second Normal Form: every non-prime Attribute in a Relation is fully functionally dependent on the primary key
3. Third Normal Form: requires that no nonprime Attribute has transitive dependency, meaning each FD X->Z and Z-> Y. It also must maintain the lossless join property and dependency preservation property
   - The lossless join property ensures that any instance of the original Relation can be decomposed into smaller Relations with no loss of information
   - The dependency preservation property ensures that after the Relation is decomposed each Functional Dependency still holds

To convert Relations into their normal form one can follow the steps provided in *Table 1. Summary of Normal Forms Based on Primary Keys and Corresponding Normalization.* The code provided by Professor Louis Nel, handles the normalization contains the code to maintain lossless join property and dependency preservation property.

Table 1. Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

Normal Form	Test	Remedy
First (1NF)	Relation should have no multivalued Attributes or nested Relations	Form a new Relation for each multivalued Attribute or nested Relation
Second (2NF)	For Relations where primary key contains multiple Attributes, no nonkey Attribute should be functionally dependent of the primary key	Decompose and set up a new Relation for each partial key with its dependent Attribute(s). Make sure to keep a Relation with the original primary key and any Attributes that are fully functionally dependent on it
Third (3NF)	Relation should not have a nonkey Attribute functionally determined by another nonkey Attribute (or by set of nonkey Attributes). That is, there should be no transitive dependency of a nonkey Attribute on the primary key.	Decompose and set up a Relation that includes the nonkey Attribute(s) that functionally determine(s) other nonkey Attribute(s).

5. ER Mapper App

ER Mapper is an android app that allows the user to create an ER Diagram and generate a relational schema in third normal form in order to create a database. Below are descriptions of the work required to complete the project including: work schedule, function requirements, use cases, and system models that explain how the system is constructed and function.

5.1 Work Schedule

Table 2. Expected and Final Work Schedule below displays the work schedule to complete this project, with expected and final dates.

Table 2. Expected and Final Work Schedule

Objective	**Estimated Time**	**Due Date**	**final date**
Research	1 week	Sept 3	Sept 2
Identify all functional Requirements	2 days	Sept 3	Sept 2
Create Structure for ER Diagram including a user interface	3 weeks	Sept 30	Sept 20
Create structure + objects for Functional Dependencies	1 week	Oct 10	Oct 5
Write + Submit Mid-term Report	1 weeks	Oct 30	Oct 30
Add complex Relationships	1 week	Nov 10	Nov 8
Apply rules to convert ER to FD for complex Relationships	3 weeks	Nov 10	Nov 14
Implement provided program for FD to DB	1 week	Dec 1	Nov 20
Write + Submit First Draft report	2 weeks	Dec 1	Dec 1
Testing and review	On going	Dec 14	Dec 15
Submit Final Report	2 weeks	Dec 15	Dec 15

5.2 Functional Requirements

The following are the functional requirements of the ER Mapper system. Each functional requirement describes in *Table 3. Functional Requirements* a set of behaviors that can be performed by the user and the system to create and map and normalize an ER Diagram.

Table 3. Functional Requirements

F-01. User must be able to create/delete Entities
a. Entities require a primary key, which is a unique name
b. In the ER Diagram the Entity will be represented by a square
F-02. Users must be able to create/remove Attributes
a. Attributes must belong to an Entity
b. Attributes can be composite or multivalued
c. In the ER Diagram the Attribute will be represented by an oval
d. The Primary Attribute will be the Primary key of the Entity, depicted with an underline
F-03. Users must be able to create/ remove Relationships
a. Connections can be between 2 Entities
b. Connections can be between an Entity and an Attribute
c. Connections between 2 Attributes show composite or multivalued Attributes
d. Connects will be depicted with a Line
e. Relationships will be depicted with a diamond
F-04. Users must be able to select a Attribute, Entity or Relationship and move it around the screen
F-05. The system must create an ER Diagram that contains a set of Entities with its Attributes and connections
F-06. The system must convert the ER Diagram into a relation schema
a. The system must convert all Relationships to binary Relationships, and then convert each Entity to a Relation
F-07. The system must identify all Functional Dependencies
F-08. The system must normalize the Relations into Third Normal Form and maintain lossless join property and dependency preservation property.
F-09. The system must create a Relational database based on the normalized Relations.
F-10. The user must be able to save their diagram in XML format
F-11. The user must be able to save the created SQLite database
F-12. The system must be able to handle and inform users of errors without crashing

5.3 Use Case Models

The following section describes the ER Mapper Program use cases. Each use case describes the behavior and functionality as modeled by *Figure 4. High Level ER Mapper Use Case*. The system has only one actor who has an option to create a diagram or normalize a diagram.

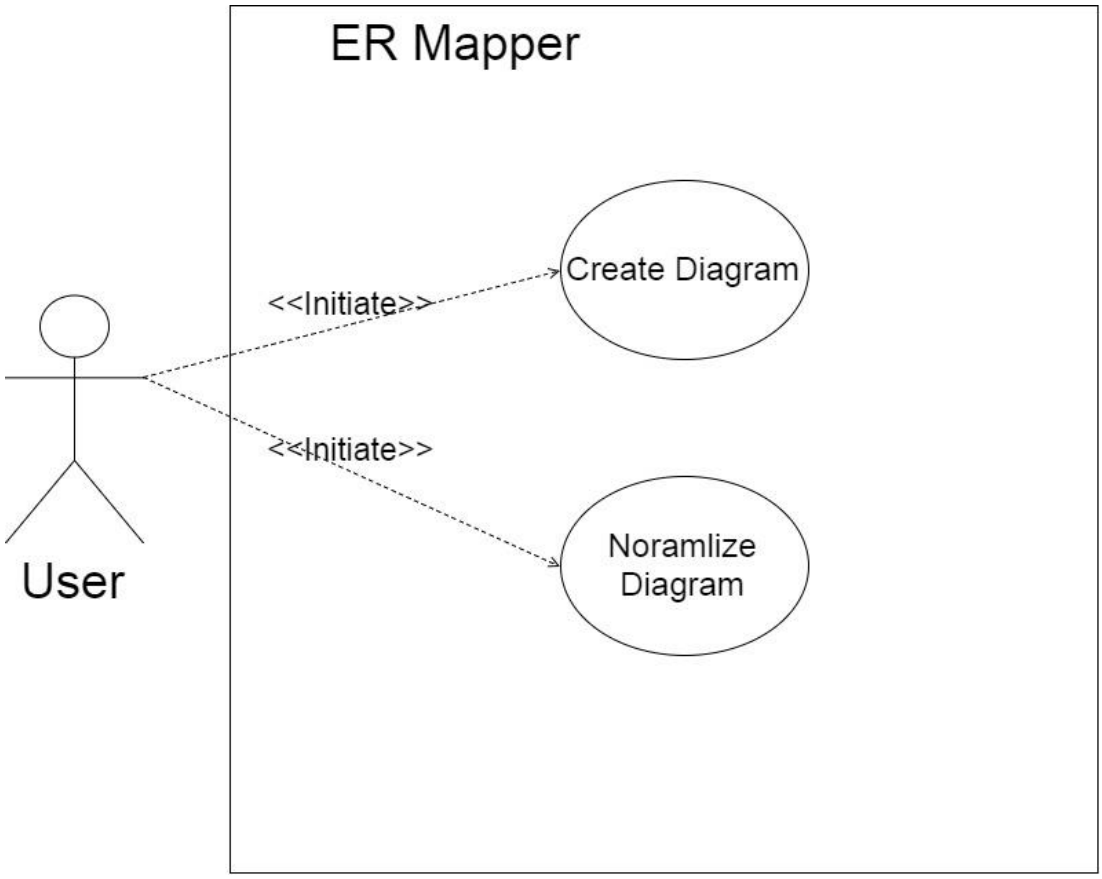

Figure 4. High Level ER Mapper Use Case

The create diagram use case describes the functionality of the system regarding creating and drawing a diagram, as shown in *Figure 5. Create Diagram Use Case.*

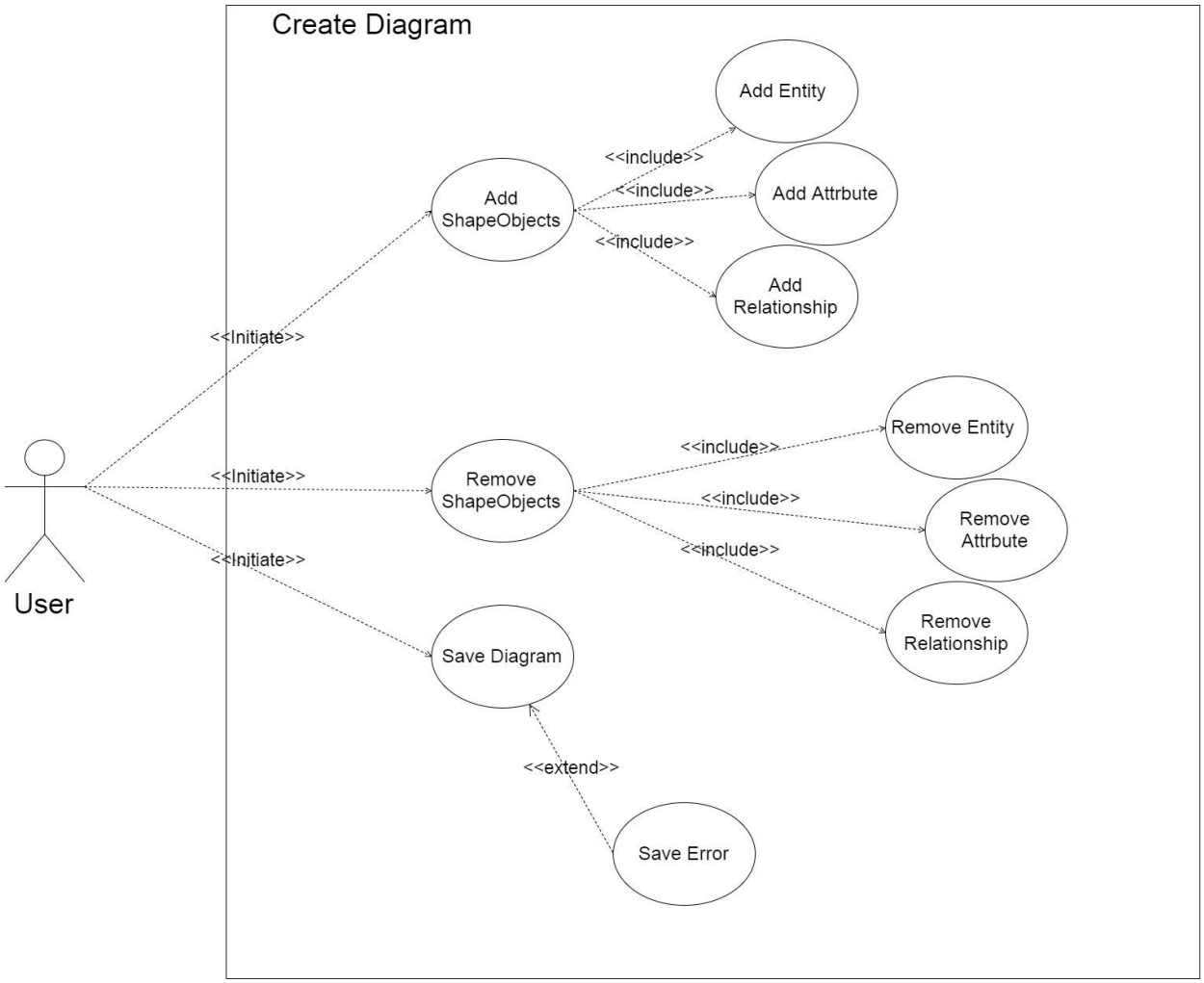

Figure 5. Create Diagram Use Case

The program also allows for a diagram to be normalize as shown in *Figure 6. Normalize Use Case*. In this process it takes the diagram and maps it to Relations to create a RelationSchema, then takes the schema and places it into third normal form. Placing the schema into third normal form ensures that the database will have a good design by removing any redundant or trivial information and meets all Functional Dependencies.

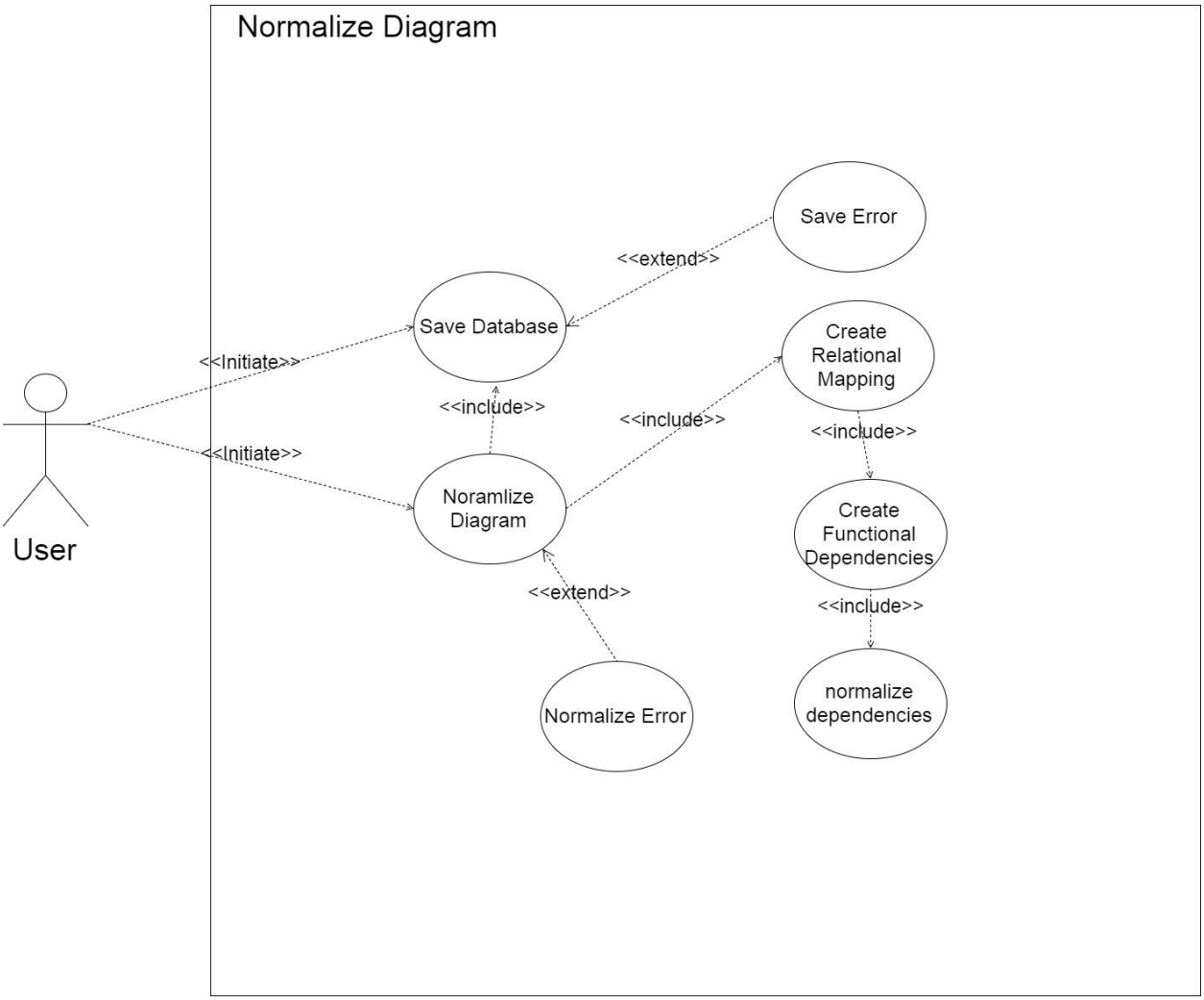

Figure 6. Normalize Use Case

The use case diagrams provide a list of use cases which can be used to describe the behavior and control flow of the system.

Table 4. High Level Use Case Descriptions walk through each use case identifying the main actors, the system responses/states, flow of events and traceability back to the systems functional requirements.

Table 4. High Level Use Case Descriptions

Use Case Identifier	UC-1
Name	ERMapper
Participating actors	User

Flow of events	1. the user selects to create a diagram a. the system will draw a blank canvas and allow the use to create the diagram 2. the user selects to create the diagram **Include Create Diagram** 3. the user selects to normalize the diagram **Include Normalize Diagram**
Entry conditions	User selects Create new diagram
Exit conditions	User creates a diagram or exits the system
Traceability	F-01, F-09
Use Case Identifier	UC-2
Name	Create Diagram
Participating actors	User
Flow of events	1. if the user selects to create an object a. The system creates a new object of the format selected. **Include addEntity, AddAttribute, AddRelationship.** Or the user selects to remove an object 2. the user clicks an object a. the system removes the object corresponding to theobject clicked. **Include removeEnitty, removeAttribute,** removeRelationship. 3. the user selects to save the diagram a. The system saves the diagram. Include **save Diagram**
Entry conditions	User selects create Diagram
Exit conditions	User selects to normalize the diagram, or exits the system.
Traceability	F-01, F-05, F-06

Error! Not a valid bookmark self-reference. depicts the control flow and traceability of how a diagram gets created using the buttons to create objects.

Table 5. Create Diagram Use Case Descriptions

Use Case Identifier	UC-3
Name	Normalize Diagram
Participating actors	User
Flow of events	1. if the user selects to normalize the diagram a. The system creates a RelationSchema. **Include Create Relational mapping.** b. The system gets a normalize error.
Entry conditions	User selects create Diagram
Exit conditions	User selects to normalize the diagram, or exits the system.
Traceability	F-09
Use Case Identifier	UC-4
Name	addEntity
Participating actors	User

Flow of events	1. the user clicks the Entity button a. the system creates a new Entity object and draws it to the screen, and adds it to the ShapeObjects list.
Entry conditions	User selects create an Entity
Exit conditions	Entity object is drawn to the screen.
Traceability	F-02
Use Case Identifier	UC-5
Name	removeEntity
Participating actors	User
Flow of events	1. the user selects delete and clicks on an Entity object a. The system searches the list of drawn objects for the selected Entity If the Entity is not part of a Relationship a. The sub objects are added to the ShapeObject list b. The Entity is removed from the shape object list If the Entity is part of a binary Relationship a. The system adds the sub objects of the selected Entity to the drawnshapes list. The system adds the other Entity to the ShapeObjects list. The Relationship is removed from the ShapeObject list If the Entity part of a ternary or n-ary Relationship a. The Entity sub objects are added to the ShapeObject list b. The Entity is removed from the Relationship c. The system redraws the diagram
Entry conditions	User selects delete an Entity
Exit conditions	Object deleted and diagram is redrawn.
Traceability	F-02
Use Case Identifier	UC-6
Name	addAttribute
Participating actors	User
Flow of events	1. the user selects Attribute button a. the system creates a new Attribute object and draws it to the screen and adds it to the shape objects list
Entry conditions	User selects create an Entity
Exit conditions	The Attributed is added and diagram is redrawn
Traceability	F-03
Use Case Identifier	UC-7
Name	removeAttribute
Participating actors	User
Flow of events	1. The user selects delete and clicks on an Attribute object a. The system searches the list of drawn objects for the selected Attribute b. The system takes any sub objects of the Attribute and adds it to the shape objects list, and removes the Attribute object

	from the drawn list c. The system redraws the diagram.
Entry conditions	User selects to remove an Attribute
Exit conditions	The Attribute is removed, and diagram is redrawn
Traceability	F-03
Use Case Identifier	UC-8
Name	addRelationship
Participating actors	User
Flow of events	1. User selects Relationship button a. The system creates a new Relationship object 2. The user clicks one object a. The system sets the center of the object to the initial position, and draws a line from there that follows the mouse. 3. The user clicks a second object a. The system sets the center of object as the second coordinates of the line. If the Relationship is between an Entity and an Attribute a. the system adds the Attribute to the Entity Attribute list, and removes the Attribute from the ShapeObjects list. If the Relationship is between two Attributes a. The system adds the second Attribute to the Attribute list of the first Attribute. If the Relationship is between two Entities b. The system adds both Entities to the Relationship object. c. The system creates cardinalities for each Entity d. The Relationship is added to the general ShapeObjects list e. Each Entity is removed from the general ShapeObjects list If the Relationship is between an Entity and an existing Relationship a. The Entity is added to the Relationship object list b. The Entity is removed from the ShapeObjects list c. The Entity gets a cardinality object If the Relationship is between an Attribute and a Relationship a. The system adds the Attribute to the Relationship b. The system redraws the diagram.
Entry conditions	User selects create a Relationship
Exit conditions	The diagram is redrawn
Traceability	F-04
Use Case Identifier	UC-9
Name	removeRelationship
Participating actors	User
Flow of events	1. User delete and clicks a Relationship object a. The system adds all sub objects of the Relationship to the ShapeObjects list. b. The system removes the Relationship from the ShapeObject

	list if it was binary.
Entry conditions	Relationship is removed, and diagram is redrawn
Exit conditions	The diagram is redrawn.
Traceability	F-04

Table 6. Normalize Diagram Use Case Descriptions depicts the control and traceability of the normalization process once the user has selected to normalize the diagram.

Table 6. Normalize Diagram Use Case Descriptions

Use Case Identifier	UC-10
Name	Save Diagram
Participating actors	User
Flow of events	1. User selects save a. The system saves the diagram in xml format b. The system displays a pop up with a success or error notification
Entry conditions	The user selects save
Exit conditions	The user presses ok on the notification.
Traceability	F-11
Use Case Identifier	UC-11
Name	Create Relational Mapping
Participating actors	User
Flow of events	a. The system takes Relationships in the diagram and applies the ER-> Relational mapping rules to create all Entities b. The system creates a Relation for each Entity If the Entity is not weak a. adds the primary Attributes of the Entity to the primary key, and all Attributes to the candidate keys. If the Entity is weak a. Adds the primary key of the strong Entity to the weak Entity primary key. Adds all Attributes to the candidate keys. If the Entity has a multivalued Attribute a. The system creates a new Relation with the Attribute containing the values list as the primary key b. All Attributes in the values list are added to the candidate keys list c. The system removes any temporary Attributes d. The system normalizes the schema. Include create **Functional Dependencies**
Entry conditions	The system creates a new RelationSchema
Exit conditions	The RelationSchema is created

Traceability	F-07
Use Case Identifier	UC-12
Name	Create Functional Dependencies
Participating actors	User
Flow of events	a. The system creates a new dependency set b. for each Relation in the schema the system adds the primary key to the left hand side, and the Attributes to the right hand side c. The system checks if the Functional Dependency is trivial if it is not trivial, the Functional Dependency is added to the dependency set d. the system performs normalization on the dependency set **include Normalize dependencies**
Entry conditions	The system created the RelationSchema
Exit conditions	All Functional Dependencies are created
Traceability	F-08
Use Case Identifier	UC-13
Name	Normalize dependencies
Participating actors	User
Flow of events	a. The system finds the minimal cover of the dependency set b. The system finds all candidate keys of the table c. The system places any Attributes not in the set in a table of their own. d. If none of the tables created contain a candidate key, then a new table is created with the candidate key e. The system removes any redundant tables
Entry conditions	The system creates a dependency set
Exit conditions	The dependency set is normalized
Traceability	F-09
Use Case Identifier	UC-14
Name	Save Database
Participating actors	User
Flow of events	1. The user clicks create database a. the system opens connection to sql b. the system creates a new database c. the system adds each Relation to the database d. the system displays a success/error prompt
Entry conditions	The user selects create database
Exit conditions	The user accepts prompt notification
Traceability	F-10, F-11
Use Case Identifier	UC-15
Name	Save Error
Participating actors	User
Flow of events	a. the system gets a file not found or database error

	b. the file is not created or saved **c.** the system displays an error prompt, **extends Save Diagram, Save Database**
Entry conditions	A save error occurs
Exit conditions	The user accepts prompt notification
Traceability	F-12
Use Case Identifier	UC-16
Name	Normalize Error
Participating actors	User
Flow of events	a. the system gets an exception when trying to parse the ER Diagram, **extends normalize diagram** b. the system displays an error prompt
Entry conditions	A normalization error occurs
Exit conditions	The user accepts prompt notification
Traceability	F-12

6. Object Models

The following diagrams are object models that represent the systems object models. They are packaged into two packages: Components and Logic, shown in *Figure 7. High Level UML Models.* The components package contains all the classes that are objects of the system, these classes include anything required to build an ER Diagram or a relational schema. The logic package pertains to all classes that control the behavior of the components, including the activity classes.

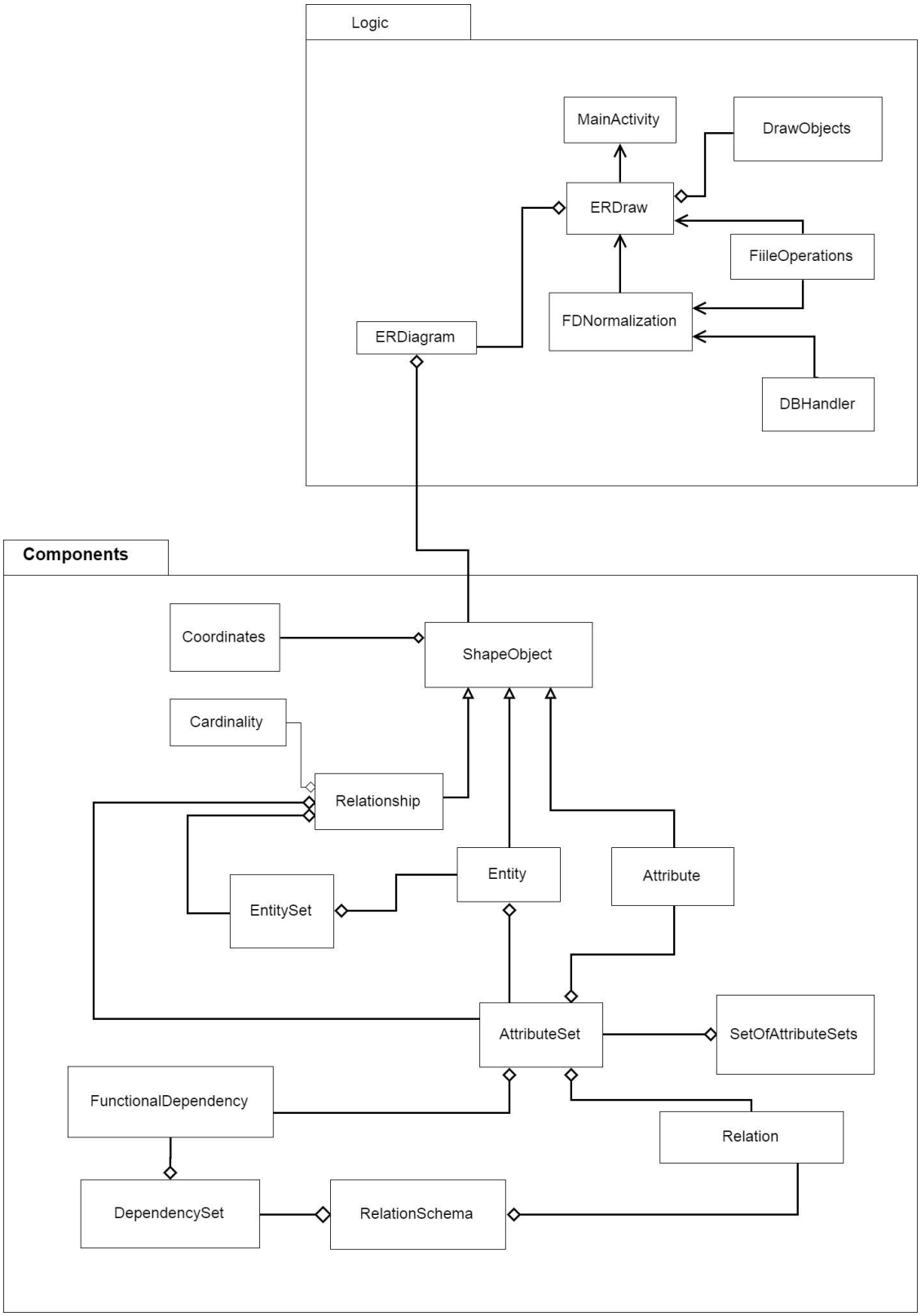

Figure 7. High Level UML Models

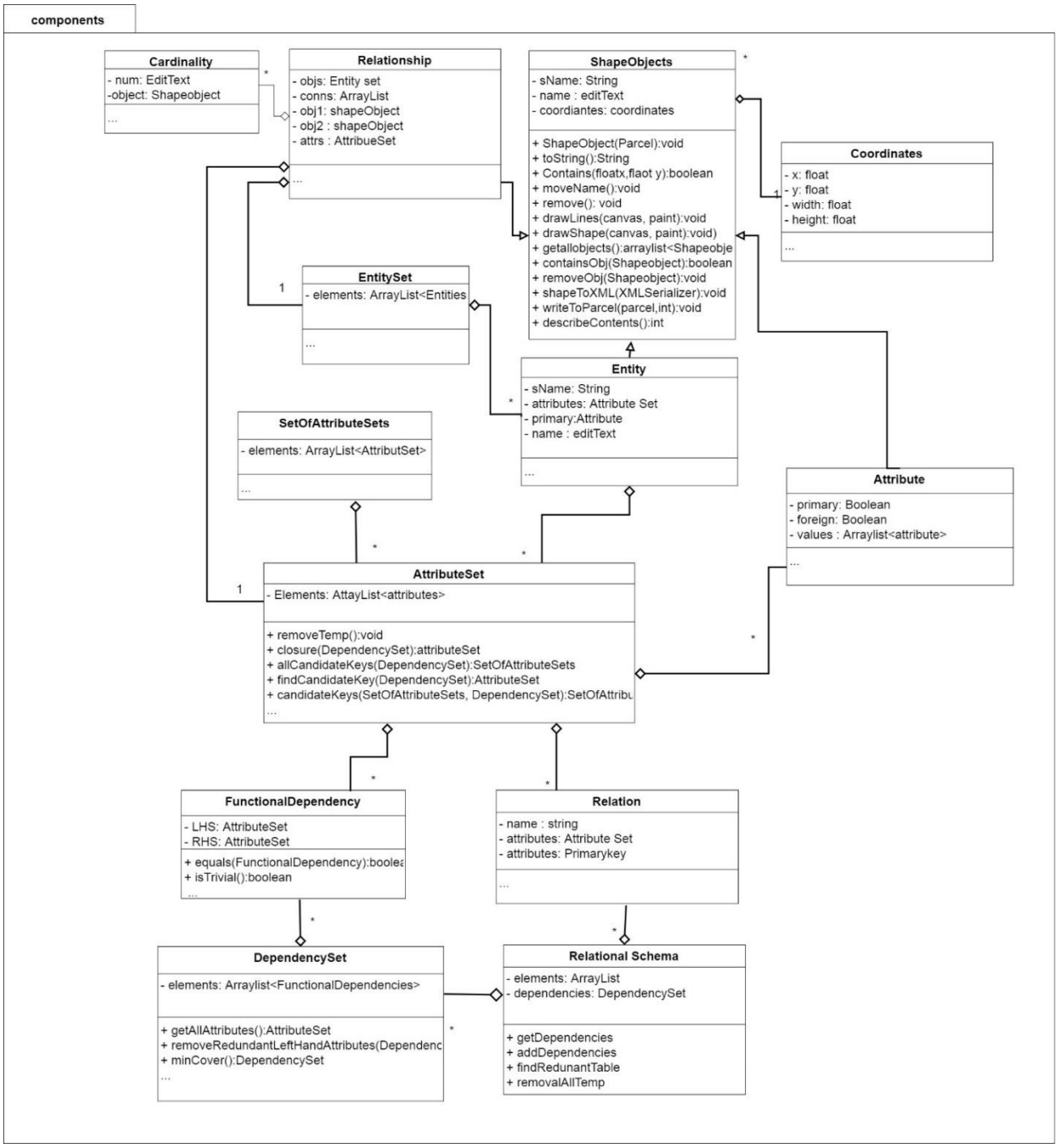

Figure 8. Components UML Model

6.1 Component Classes

The component classes are modeled in *Figure 8. Components UML Model* and make up all nonlogic classes that represent an idea that needs to be modeled by the ER Diagram. For better readability the Components UML diagram includes all class objects but excludes simple functions such as setters and getters. The ShapeObjects class, which is abstract represents shows all functions that are inherited by the subclasses.

6.1.2 ShapeObjects

An ER Diagram is modeled using symbols, called ShapeObjects in the ER Mapper system. A ShapeObject is an abstract class that allows for the diagram to store each ShapeObject and offers for a single interface to be used for each ShapeObjects. Within this class there are several abstract functions that get overwritten by the sub classes defining specific behavior, such as drawLines(), drawShapess(), removeObj(), getallObjects(), that cover the functionality of each ShapeObject type. Each of these objects contains a name, edit text and coordinates which allow form them all to be drawn to the canvas. Each edit text has an event listener attached to it, making it possible for the user to change any object name. ShapeObject extend Parceable class so that objects can be passed between android activities. Parceable works, by decomposing an object into a parceable object using the wrtiteToParcel() function, and it then uses the readsFromParcel()function to calls a creator class to reconstruct the object from the parcel.

6.1.3 Entity/Entity sets/Weak Enitites

As explained in the research section, an Entity represented a Relational table, which models some aspect of the real world. An Entity is a subclass of a ShapeObject. An Entity is represented as a square, and contains a solid line to connect its Attributes. An Entity contains a list of Attribute, and a Boolean which is true if weak, false otherwise. A weak Entity is drawn as a square outlining the original square, and two solid lines connecting the Relationship to represent total participation in the Relationship. An Entity set, contains a list of unique Entities.

6.1.4 Attribute/Attribute sets/SetOfAttributeSets

An Attribute is also a subclass of ShapeObject and its symbol is an oval. Primary Attributes are used to uniquely identify an Entity. An attribute object represents a property of either an Entity or Relationship. In a Relational database where every Entity is a table, each Attribute identifies a

column. A primary key is an Attribute that must be unique for ever row in the table. Within the ER Mapper system to create a primary Attribute, a user can double tap the attribute they want to set it to primary or remove the primary. A primary key attribute is identified by underlining its name in the ER Diagram. A foreign key is a primary key of a weak Entity that references the primary key of another table. The foreign key will be identified with a dashed underline of the name. Along with being primary and foreign, an Attribute can be multivalued. A multivalued attribute can be broken down into separate components, such as a date of birth, which can be decomposed into year, month, day. If the Attribute is multivalued, the sub Attributes are stored in a list.

An Attribute set, contains a list of unique Attributes. When a parceable creates an AttributeSet it calls the creator for an Attribute, which causes the creator to get stuck in a loop, therefore all multivalued Attributes are stored in an array list instead of an Attribute. Another issue that occurs do to the parceable implementation, is that each time it calls a creator a new instance is created. Therefore; if the same Attribute has two references, the parceable creates two new references for it. Then when checking if an Attribute already contains that Attribute, it does not recognize the objects as the same. As a work around, the Attribute set checks if the names are equal. It is important to note that two Attributes with the same name cannot be added to any Attribute set. When any Attribute is created it is called "object" + count, where the count is incremented for every Attribute created making the name unique for every Attribute, so that it can be added to an Attribute set before given a unique. The SetOfAttributeSets class holds a unique set of Attribute sets.

6.1.5 Relationship/Cardinality

A Relationship is another subclass of the ShapeObject which is modeled a line connection between two or more Entity objects. If the Relationship is an identifying Relationship, it gets a diamond with a diamond outline, and has two solid lines to show total participation. A Relationship object may also contain an Attribute set, representing any properties of the Relationship. An Entity set is used to stores all Entities participating in the Relationship. Each Entity in the Relationship can contain a different cardinality in the Relationship. Therefore, for each solid line a cardinality object is drawn. A cardinality object contains an edit text, initially set to 1, and can be modified by the user to model 1:1, 1: N and M : N Relationships.

6.1.6 Relations/Relation Schema

The Relation class represents an Relational table, derived from an Entity object and its Attributes and Relationships. A Relation contains an AttributeSet of nonprimary keys, an AttributeSet of primary keys and a name. A Relation is created by iterating through all Entity Attributes, adding each to the Attribute set, checking if the Attribute is primary or foreign, and adding it to the primary set if true. The primary key should be a subset of the Attribute set. A relational schema, represents a unique set of Relations. When creating a RelationSchema, the program looks at every object in the general ShapeObject list representation of the ER Diagram, and maps it to Relations. The rules for ER to Relational mapping rules from *Fundamentals of Database Systems* is used as follows:

1. For all 1:1 Relationships do
   - Choose one of the Relations-say S-and include a foreign key in S the primary key of T
   - It is better to choose an Entity type with total participation in R in the role of S
2. for All 1:N Relationships do
   - create a Relation S that represent the participating Entity type at the N-side of the Relationship type
   - Include as foreign key in S the primary key of the Relation T that represents the 1 side of the Relationship type
   - Include any simple Attributes of the 1:N Relation type as Attributes of S
3. For all M:N Relationships do
   - For each regular M:N Relationship type R, create a new Relation S to represent R
   - Include as foreign key Attributes in S the primary keys of the Relations that represent the participating Entity types; their combination will form the primary key of S
   - Also include any simple Attributes of the M:N Relationship type (or simple components of composite Attributes) as Attributes of S
4. For multivariable Attributes
   - For each multivalued Attribute A, create a new Relation R
   - This Relation R will include an Attribute corresponding to A, plus the primary key Attribute K-as a foreign key in R-of the Relation that represents the Entity type of Relationship type that has A as an Attribute
   - The primary key of R is the combination of A and K. If the multivalued Attribute is composite, we include its simple components

Once the ER Diagram has been mapped to its relational schema, the program can check for constraints and remove any trivial or redundant information.

6.1.7 Functional Dependencies / Dependency Set

A Functional Dependency represents the constraints on a Relation. A Functional Dependency, contains a left had side with all primary and foreign Attributes of a Relation, as well as a right-hand side with all nonprimary Attributes of a Relation. A Dependency set contains a list of unique Functional Dependencies.

6.2 Logic Classes

All logic classes are shown in *Figure 9. Logic UML Model* the following sections again explains and describes what each class is and how they work.

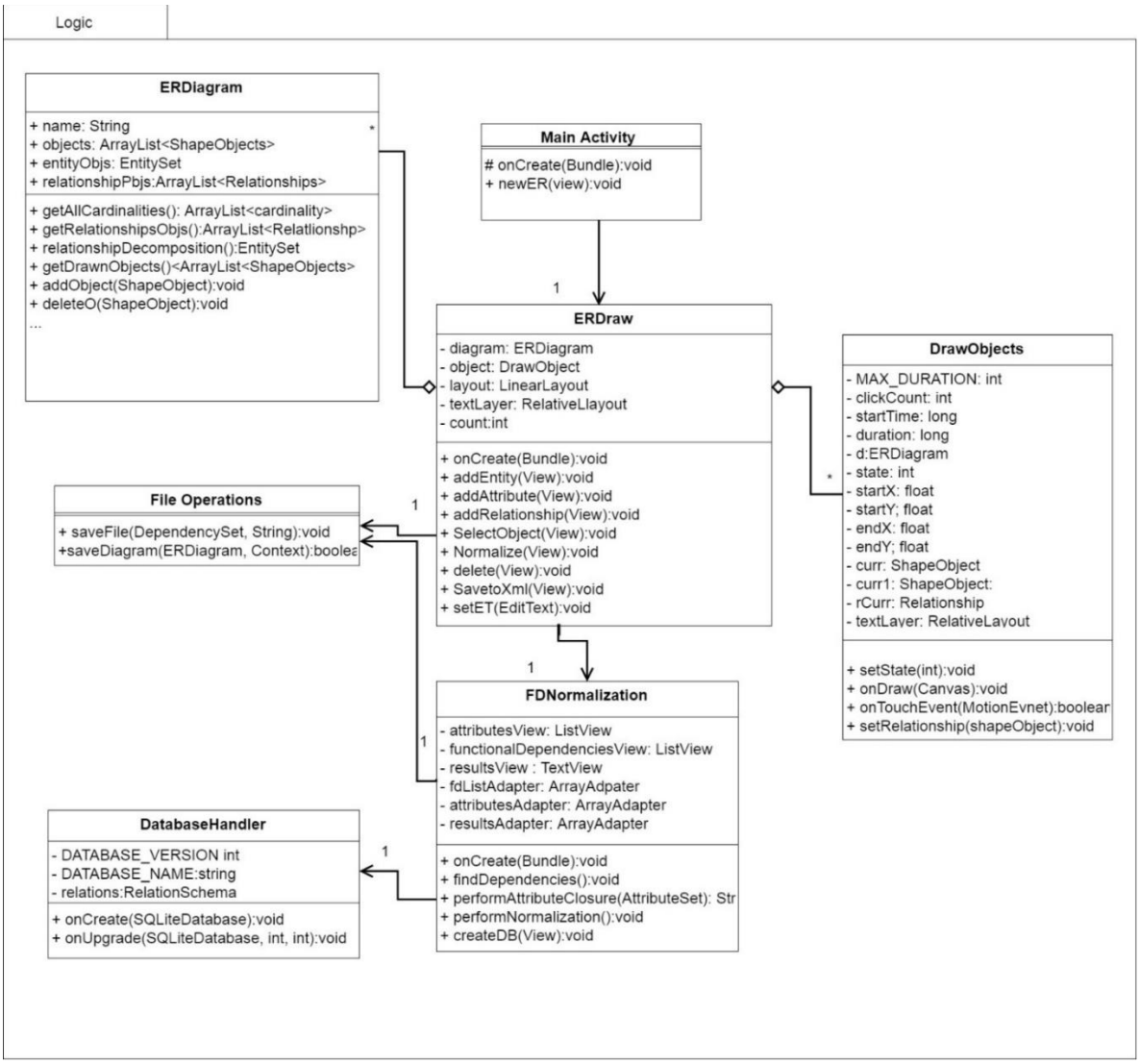

Figure 9. Logic UML Model

6.2.1 Activity Classes

MainActivity, *ERDraw* and *FDNormalizations* are each activity classes. According to Android studio API an activity class represents a single screen with a user interface. The MainActivity class is the first screen that appears when the user launches the app, it launches to a screen where the user can select to create a new ERDiagram. From there, the main activity launches the ERDraw activity and passes it a new ER Diagram object. The ERDraw activity creates the interface for the blank canvas. It contains several buttons at the bottom of the screen which allows the user to draw object, adding/removing/editing the ER Diagram object. The ERDraw activity also contains a Realative Layout where all objects are added to, and a Relative Layout where all edit text objects are stored. This allows for all edit text objects to be placed on top of the linear layout so that they are always visible. The user may also select the normalize button which launches the FDNormalization activity, which creates a new interface that displays the RelationSchema details. The FDNormalization activity has several text view objects within a scroll box that display all information in the system, along with a create diagram button, which will create SQLite database and save it locally to the android device.

6.2.2 ER Diagram

The ER Diagram class is used to represent an ER Diagram that will be drawn to a blank canvas. When an object is drawn it is added to an arraylist of ShapeObjects. As each object forms Relationship Attributes and Entities are added to their corresponding owner, then removed from the general ShapeObject list. In doing this it ensures if there exist any objects that are not in a Relationship, they can still get drawn, but removes storing duplicate information. Since the object is removed from the general list, the Draw class searches each object for any of its members to draw instead of just looking at the list. For example, suppose there exist an Entity and Attribute object with no Relationship. Then each object will be stored in the general ShapeObject list.

Suppose a Relationship is created between he Entity and Attribute. The Attribute gets added to the Entity Attribute list, along with this a new Relationship is created, storing the Attribute and Entity as objects 1 and 2. Then the Attribute and Entity are removed from the ShapeObject list and the Relationship is added. To draw this Relationship, the Relationship drawLines() function is called to draw the lines between the Entity and Relationship, then the draw shapes function is call to draw the actual Entity and Attribute shapes.

The function relationshipDecomposition() is used to look at the diagram and organize it in terms of its Entities. It starts by looping through every Relationship checking if it is not binary and decomposing it to a binary Relationship, using the steps mentioned earlier. In the conversion, several Entity objects will be created. When these Entities are created they are given the Attributes of their connecting Entities**,** in some scenarios this may create an Entity that is trivial, i.e. contains a Functional Dependency X-> Y where Y is a subset of X. To bypass this, it creates a temporary Attribute with a name of "-1", which allows for the Entity to be considered valid while it gets mapped to a Relation. Once the relationshipDecomposition() function is finished the system will have properly identified all Entity objects and Attributes, and the Entities can be mapped to their Relations using the steps described before in the section on Relation/ RelationSchema. Once the RelationSchema is created, there is a function removeTemps() that removes the temporary Attributes from the Schema.

6.2.3 Draw Objects

The ERMapper starts by creating a blank canvas that allows users to draw an ER Diagram. The general drawnObjects() function from the ER Diagram class is used to identify which can be Entities, Attributes or Relationships, with their coordinates to be drawn to the screen. The main page of the application seen in *Figure 10. ER Mapper Drawing* shows the canvas of the page, and in the red box identifies the buttons that user can use in order to create new objects.

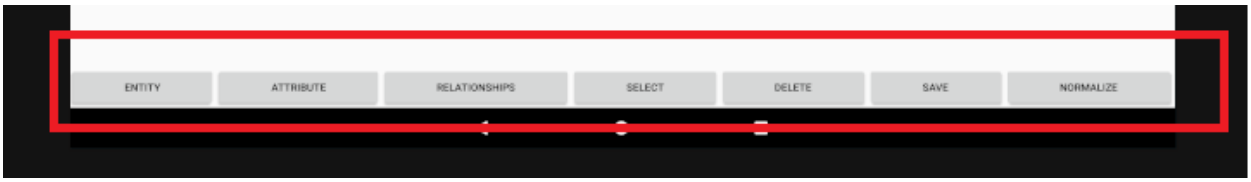

Figure 10. ER Mapper Drawing

Once the user has selected a button, the corresponding object is created. If the user selected to create a Relationship the system creates a line that follows the mouse to connect to objects. If the Relationship is valid then it gets added to the object list. The program searches the ERDiagram list of objects and draws the correct shape, based on the object type onto the screen at the correct

coordinates. The canvas also has a motion event listener which activates when the user clicks the screen. It checks if the user has clicked a coordinate that is inside a shape and then allows the user to move that object around the screen, if the object has any Relationships it will also update those coordinates. To create key Attributes a user can double click on the Attribute they want to make key, and an underline will appear. To make a weak Entity the user can double click an Entity. Once the user has completed has completed their drawing they can click normalize to normalize the objects and create a Relation diagram. *Figure 11. Completed ER Diagram* below is an image of a complete ER Diagram.

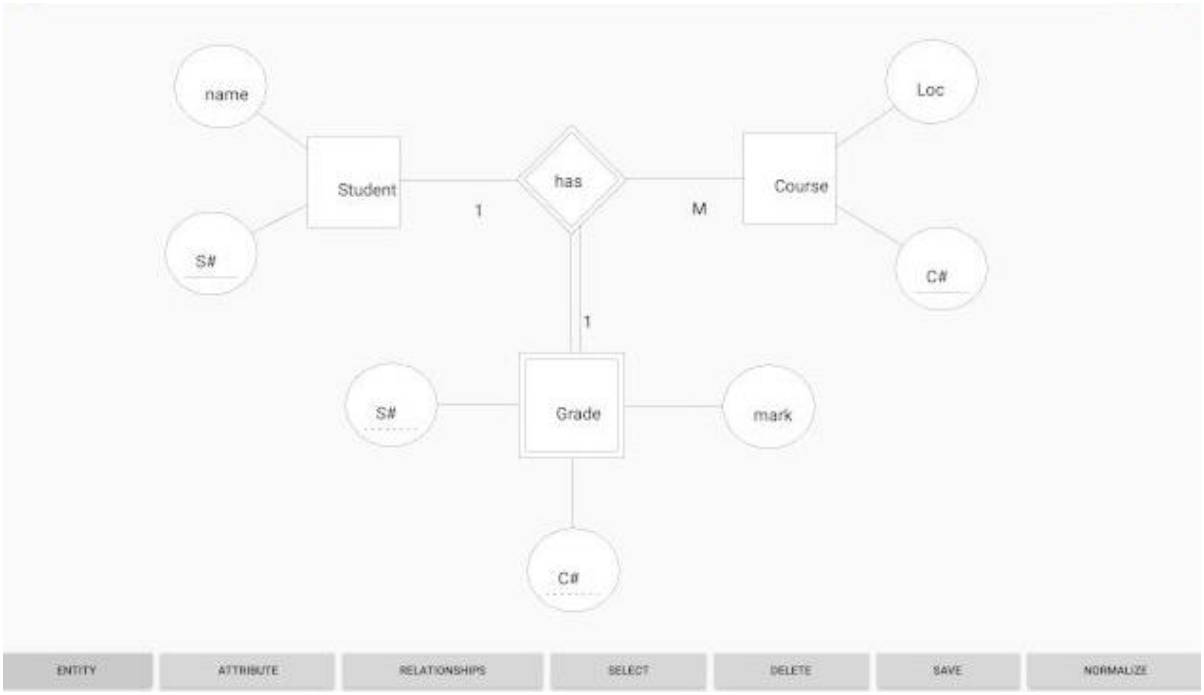

Figure 11. Completed ER Diagram

6.2.4 FD Normalization

To normalize the diagram, one must press the Normalize button from the diagram. The ERDraw class will cause the FDNormazliation activity. This class will parse through all of its Relationships and convert them to binary Relationships, and then create an arraylist of Entities which entirely covers the scope of the diagram in terms of Entities, removing redundancy's and any information that is not relevant to creating a database such as coordinates. It will display any relevant information in system to the screen and offer an object for the user to create a database.

Figure 12. Normalized ER Diagram is an example of the FDNormalization activity. In the results section is a scroll box where the following information is displayed:

1. All Attributes
2. All Functional Dependencies
3. Minimal cover
4. Checks that the minimal cover is equivalent to all original Functional Dependencies
5. All Candidate keys of a table
6. Lossless-join and Dependency Preserving, 3NF tables
   - This are the tables that will be used to create the database

In order to get all the Attributes, Functional Dependencies and results, the class takes the created RelationSchema, normalizes it and performs Attribute closures. The FDNormalization class calls the relationshipDecomposition() function which as mentioned before, decomposes all Relationships to binary Relationships and returns a list of all the new and old Entities. The system can then map those Entities to Relations and create a new RelationSchema by using the following steps from the book *Fundamentals of Database Systems*:

1. for all Regular Entity types, assign a Relation, pick a primary key
   a. if the primary key is a complex Attribute: all Attributes will be included
   b. if an Attribute is complex, create a new Relation
2. for all Weak Entities, create a foreign key that references all Primary keys of its Strong Relation

Once the RelationSchema is mapped each Functional Dependency is identified such that any primary key in a Relation must functionally determine all other Attributes in the Relation. To do this the program sets the primary key to the left-hand side of the Functional Dependency and all other Attributes to the right-hand side. From the Functional Dependencies and primary keys of each Relation, the system can use the performNormalization() function which applies the steps that were mentioned earlier, to normalize the RelationSchema into Third Normal Form. The code that handles checking lossless-join and dependency preservation properties was provided by Professor Louis Nel. The algorithm itself was based on the four-step algorithm 16.6 presented *Fundamentals of Database systems*, which decomposes a set of Attributes with respect to Functional Dependencies F:

1. find a minimal cover Fm of F
2. for each left hand side X of FD in Fm create with columns X U A1 U A2 U ...An where X->A1, X->A2,... X->An are all the dependencies in Fm with left hand side X
3. if none of the tables created in Step 2 contains a candidate key for the universal Relation consisting of all the Attributes, then create a table consisting of a candidate key remove redundant tables. If any table is a projection of another (has all its columns appearing in another table), it is removed table

Figure 12. Normalized ER Diagram, shows the screen that gets displayed after the normalization process which displays the System Results. The results screen prints out information relevant to the process including, finding the minimal cover of the Relations, and finding all candidate keys which prove that the lossless join property and dependency preservation property hold still.

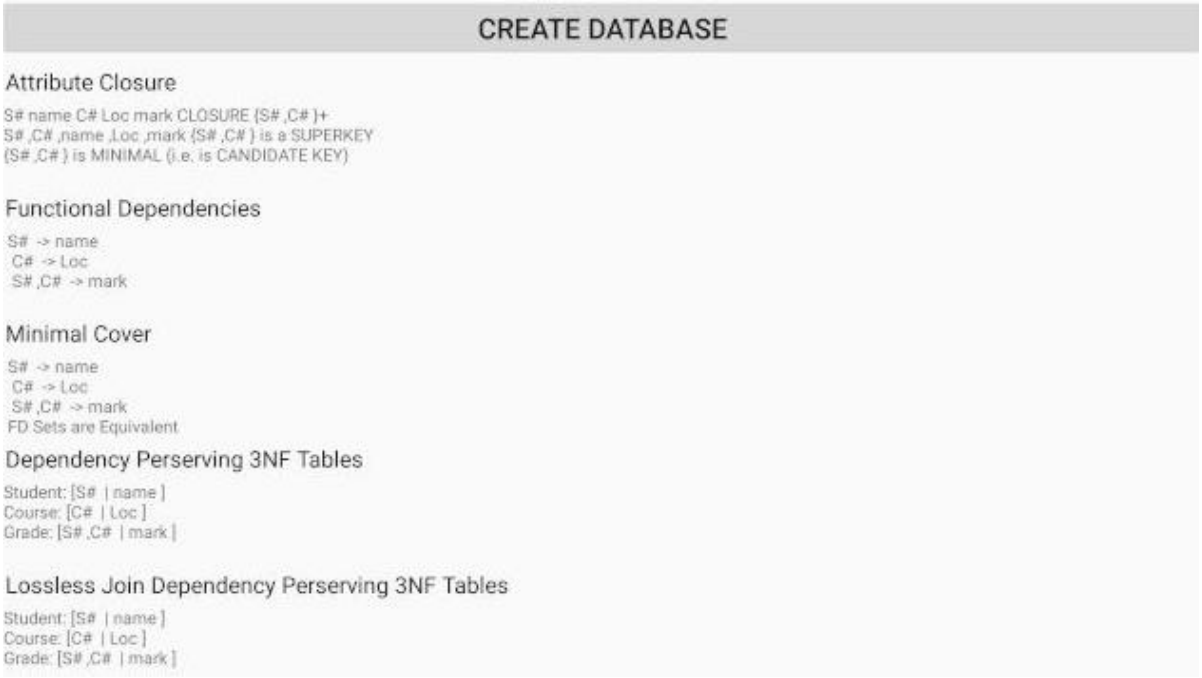

Figure 12. Normalized ER Diagram

From the FDNormalization activity page users my select to create a database. Once the button is selected a SQLite Database is opened, and a table is created for each relation with the required columns from the relation attributes. If the attribute is a primary key, it is designated a primary

key in the relation table in the database. If the attribute is a foreign key, the program searches all relations for a primary key with the same name. As the program finds the identifying relation by matching the name of the primary key to the foreign key, this creates a constraint on the system that the primary key of the identifying relation has the same name as the foreign key of the weak entity key attribute. From here the program has everything it requires to create the database, upon which a user gets a notification that the database was successfully/ or unsuccessfully created.

7. Results

To ensure that the system works correctly, and its actual outputs match its expected outputs, there were a series of unit tests done using unit Tests. All tests in the system display their output to the console at the bottom of screen. The logic unit test cases (**src->test**) test, that functions have the desired response of the system to ensure that the ERDiagram created and normalized. Along with the logic test cases are the diagram and database test cases which are emulated test(**src->androidTest)** that check the system User Interaction. *Table 7. Test Case Matrix* shows all tests that ran in what test suite they will run in and how they traceback to the Functional Dependencies. The tests run by **right clicking** the desired test and selecting **run.** The tests below do not cover the entire functionality of the system and more tests should occur to ensure that the relationship decomposition and normalization procedures are correct and receive the desired responses. This can occur by create several different diagrams of different varieties to test as well as by including scenario tests that consider different sequences of execution. However even without testing the entire functionality the completed tests give a good indication that the system is doing as expected.

note: that when running emulated test suites sometimes the tests will cause an error due to conflict from the previously run test suite. However, they do all run successfully independently.

Table 7. Test Case Matrix

Test Cases	Location	Traceability
addEntity ()	Logic	F-01
canRemoveWeakEntity()	Logic	F-01
CreateEntity()	Diagram	F-01
CreateWeakEntity ()	Logic	F-01
removeEntity()	Logic	F-01
removeEntityWithAttriubte()	Logic	F-01
AddAttributeToEntity()	Logic	F-01, F02
DeleteObject()	Diagram	F-01, F-02, F-03

removeEntityFromBinaryRelationship()	Logic	F-01, F-03
removeEntityFromTernaryRelationship()	Logic	F-01, F-03
addAttribute()	Logic	F-02
CreateRelationship()	Diagram	F-02
removeAttribute()	Logic	F-02
removeAttriubteFromEntity()	Logic	F-02
setPrimaryAttribute()	Logic	F-02
addRelationship()	Logic	F-03
CreateAttribute()	Diagram	F-03
SelectObject()	Diagram	F-04
BinaryDecomposition()	Logic	F-06
NormalizationTest.RelationSchema()	Logic	F-06, F-08
NormalizationTest.FunctionalDependencies ()	Logic	F-07, F-08
CandidateKey()	Logic	F-08
dp_lj()	Logic	F-08
MinCover()	Logic	F-08
NormalizationTest.GetAllEntityObjects ()	Logic	F-08
thirdNF()	Logic	F-08
CreateDB()	Database	F-09, F-11
Save()	Diagram	F-10

8. Conclusion

The purpose of this project was to create an android app that will allows uses to drawn ER Diagram and automatically generate the corresponding RelationSchema. The ERMapper program successfully meets all of its functional requirements and successfully creates a Relational database. The ER Mapper offers a simple and effortless way for database designer to quickly create a database and have an idea of how the mini-world they are modelling is related.

Some issues with the system include double clicking to create a primary Attribute or a weak Entity, as Entity and Attribute objects contain an edit text in the middle of their object, sometimes when double tapping you tap the edit text to change the name instead of creating a primary attribute or weak entity. This issue is especially prevalent if using a small android device or if you have big fingers. A way to get around this would be to use a tablet pen or making the objects larger. This shows that the small screens of android devices create a constraint on the usability of the system. Another issue with the system occurs when mapping complex relationships including only supporting 1, N or M being entered as relationship cardinality. Along with limitations to cardinality as part of the relationship decomposition new entities and relationships are created with a placeholder name, that of the relationship name, that cannot be edited, this adds some ambiguity

as the user may not be aware of what the entity or relationship is trying to model. The ambiguity also creates issues with foreign keys, as they must be named the same as their identifying key in order for the program to ensure correctness. If the primary key and foreign key do not have the same name, the system will reference the foreign key and primary key separately instead of using the foreign key to reference the primary key. Currently the system is set up that every primary key of a weak entity becomes foreign, this means that the system does not support a weak entity that has a combination of primary and foreign keys. It should also be noted that ER Diagrams themselves are ambiguous as the system does not have an understand of what everything is so while it ensures that the user meets the minimal constraints of an ER Diagram, such as an entity requires a primary attribute and weak entities require strong entities, if the user creates a diagram that incorrect the database will reflect that.

Future work for the system include performing some usability testing to find the best size and constraints for the user interface as well as testing for more complex relationships. The system could also include creating a file opener so that users can load already existing diagrams from their XML format into the system. Another downside of the system is that is works only on android devices. Android devices tend to be smaller making it difficult to draw more complex ER Diagrams, as you cannot see the whole diagram at once. Being an android app also makes it difficult to access any files that are saved. By creating the same system as web application so that it is accessible on non-android devices. As a work around for this though, the app can always be run in android studio as an emulator which allows for a bigger screen size if you do not have a large android device, and files can be transferred from the android device to your computer, so that once your database is created you can move it to a better location.

The ER Mapper app would be a great tool database designers and Professors to use explain how the database design and its normalization to process to clients and students. By providing immediate feedback the app provides faster and more efficient solution then using a pen and paper. While the app does have some limitations including size and complexity, however with some more testing and refactoring the app can be improved to offer an even larger variety of uses as well as can be converted to different platforms.

References

Activity. (2017, October 25). Retrieved November 20, 2017, from https://developer.android.com/reference/android/app/Activity.html

Elmasri, R., & Navathe, S. (2016). *Fundamentals of Database Systems*. Don Mills, Ont.: Addison-Wesley.

Nel, L. D. (2015, Fall). *Algorithmic Design with Normal Forms*. COMP 3005 Course Notes Fall 2015.

Nel, L. D. (2015, Fall). Modeling ER Features with Functional Dependencies. COMP 3005 Course Notes Fall 2015.

Nel, L. D. (2015, Fall). *Normal Forms*. COMP 3005 Course Notes Fall 2015.