Entity Relationship Diagram Mapping

by

Cheryl Dunn

100963953

Prof. Louis D. Nel

School of Computer Science

Carleton University

Ottawa, Ontario

# Abstract

ER Mapper is an android app that takes an Entity-Relationship diagram (ER diagram) map it to its relations, find functional dependencies, perform normalization and create a relational database. Users have the ability to draw and ER diagram to a canvas, such that they can create, remove and edit entity 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 and create a database

# Acknowledgments

Code was supplied by Prof. Lou Nel that allowed for Functional Dependencies to be normalize preserving lossless join and dependency preservation properties.

Table of Contents

[Abstract 2](#_Toc499313205)

[Acknowledgments 2](#_Toc499313206)

[Intro 4](#_Toc499313207)

[Thesis 4](#_Toc499313208)

[Work Schedule 4](#_Toc499313209)

[Setup 5](#_Toc499313210)

[Running the software 5](#_Toc499313211)

[Functional Requirements 5](#_Toc499313212)

[Research 6](#_Toc499313213)

[Entity Relationship Diagrams 6](#_Toc499313214)

[ER Diagram Components 7](#_Toc499313215)

[Entity objects 7](#_Toc499313216)

[Relationship 8](#_Toc499313217)

[Attribute 8](#_Toc499313218)

[Converting Relationships to Binary 9](#_Toc499313219)

[Functional Dependencies 9](#_Toc499313220)

[Use Case Diagrams 10](#_Toc499313221)

[Use Case Descriptions 12](#_Toc499313222)

[Classes 18](#_Toc499313223)

[ShapeObjects 21](#_Toc499313224)

[Entity/Entity sets 21](#_Toc499313225)

[Attribute/Attribute sets/SetOfAttributeSets 21](#_Toc499313226)

[Relationship/Cardinality 22](#_Toc499313227)

[Relations/Relation Schema 22](#_Toc499313228)

[Functional Dependencies / Dependency Set 23](#_Toc499313229)

[Activity Classes 24](#_Toc499313230)

[ER Diagram 25](#_Toc499313231)

[Draw Objects 25](#_Toc499313232)

[FD Normalization 27](#_Toc499313233)

[Normalization 29](#_Toc499313234)

[Known issues 29](#_Toc499313235)

[Results 30](#_Toc499313236)

[References 30](#_Toc499313237)

List of Figures

[Figure 1 Green Run Arrow 5](#_Toc499313238)

[Figure 2. ER diagram Shapes 7](#_Toc499313239)

[Figure 3 Weak Entity Example 8](#_Toc499313240)

[Figure 4 Degree of Relationships 8](#_Toc499313241)

[Figure 5 High Level ER Mapper Use Case 10](#_Toc499313242)

[Figure 6. Normalize Use Case 12](#_Toc499313243)

[Figure 7. HighLevel UML Models 19](#_Toc499313244)

[Figure 8. Compoenents UML Model 20](#_Toc499313245)

[Figure 9. Logic UML Model 24](#_Toc499313246)

[Figure 10 ERMapper Drawing 26](#_Toc499313247)

[Figure 11 Completed ERDiagram 27](#_Toc499313248)

[Figure 12 Normalized ERDiagram 28](#_Toc499313249)

List of Tables

[Table 1 Use Case Descriptions 11](#_Toc499239608)

[Table 2 Summary of Normal Forms Based on Primary Keys and Corresponding Normalization 22](#_Toc499239609)

[Table 3 Work Schedule 24](#_Toc499239610)

# Intro

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 data and 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, tables, queries, reports and objects. When designing a database, it is important to organize the data so that it models the data in a realistic way to support the information being collected, with reduced redundancy. For database designers to organize data they may first make a visual diagram which will allow them to identify entities, attributes and their relationships. An Entity Relationship (ER) Diagram is a widely used method for conceptualizing and visualizing the logical structure of a database. By Creating an ER Diagram and normalizing it to its functional dependencies Database Designers can easily create an accurate database for whatever data needs to be stored. ERMapper is an android app usable on any android device such as a tablet. Database designers can bring their tablet with them to meetings with the client as well as with their team. This allows for users to quickly and efficiently get an idea of what there database will look like. The ER Mapper is a program that allows users to create an ER diagram and then converts it to a relational database would help save time, ensure consistency and accuracy. This report walks through how to setup and run the ER mapper program and explain how the program talks an ERDiagram, maps it to its relations and functional dependencies to create a database.

# Setup

The following are instructions of how to install and set up the software required to run the ERMapper application in Android studio.

1. Install Android studio 3.0
   1. JRE: 1.8.0\_152-release-9159b01
   2. JVMOpenJDK 64-bit Server VM by JetBrains s.r.o
2. In Android studio go to File -> new -> import Project. A dialog will pop up, select the directory where you saved ERMapper.zip and the project will open.
3. The following are instructions on how to setup the program to run on an emulator in android studio
4. In Android studio go to **Tools** > **Android** > **SDK Manager.**
5. 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).
6. In Android studio go to **Tools** > **Android** > **AVD Manager.**
7. When the pop up screen opens select create virtual device. Go to **Tablets** > **Nexus 10** then press Next.
8. On the Next screen make sure the SDK that you choose from step 2 is selected and then click next and finish.
9. The following are instructions on how to setup the program to run on android device.
10. Enable USB Debugging on your android device
    1. Open the **Settings** app.
    2. (Only on Android 8.0 or higher) Select **System**.
    3. Scroll to the bottom and select **About phone**.
    4. Scroll to the bottom and tap **Model number** 7 times.
    5. Return to the previous screen to find **Developer options** near the bottom.

# Running the software

Now that you have imported the code and set up all the necessary software you can run the program. Press the green run arrow depicted in the image bellow, it will launch a prompt that asks you to select a device, you may choose to run on the android device or emulator and press OK. This will install the software onto your android device or launch the emulator. When it is ready the ERMapper will launch.



Figure Green Run Arrow

# Work Schedule

The table below displays the work schedule to complete this program, with expected and final dates.

Table 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 |  |
| **Testing and review** | On going | Dec 14 |  |
| **Submit Final Report** | 2 weeks | Dec 15 |  |

# Functional Requirements

The following are the functional requirements of the ER Mapper system. Each functional requirement describes a set of behaviors that can be performed by the user and the system to create and map and normalize an ER diagram.

Table List of Functional Requirements

|  |
| --- |
| 1. User must be able to create/delete *entities* |
| * 1. Entities require a primary key, which is a unique name |
| * 1. In the ER diagram the Entity will be represented by a square |
| 1. Users must be able to create/remove attributes |
| * 1. *Attributes* must belong to an *entity* |
| * 1. *Attributes* can be composite or multivalued |
| * 1. In the ER diagram the *attribute* will be represented by an oval |
| * 1. The *Primary* *attribute* will be the Primary key of the *entity*, depicted with an underline |
| 1. Users must be able to create/ remove *relationships* |
| * 1. Connections can be between2 *entities* |
| * 1. Connections can be between an *entity and an attribute* |
| * 1. Connections between 2 *attributes* show composite or multivalued *attributes* |
| * 1. Connects will be depicted with a Line |
| * 1. *Relationships* will be depicted with a diamond |
| 1. Users must be able to select a *attribute, entity or relationships* and move it around the screen |
| 1. The system must create an *ER Diagram* that contains a set of Entities with its attributes and connections |
| 1. The system must convert the ER diagram into a relation schema |
| 1. The system must identify all functional Dependencies |
| 1. The system must normalize the relations into Third Normal Form maintain lossless join property and dependency preservation property. |
| 1. The system must create a relational database based on the normalized relations. |

# Research

Within this section key ideas and concepts regarding ERDiagrams, FDNormalization and Databases will be identified, as described from the textbook *Fundamentals of Database Systems* by Ramex Elmasri & Shamkant B. Navathe.

## Entity Relationship Diagrams

ER Diagrams are a visualization of Relations in a database. Entities, modeled as squares, represent a real world concept that a Relation is, where as attributes, modelled as ovals, represent property of its corresponding relation. In a Database the Entity or Relation is the table itself, and the attributes are the columns in that table. Every entity can have multiple attributes, however it requires at least one attribute to be a key (which uniquely identifies each row in the table). In an ER diagram the 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, though at this point the program only deals with binary 1:1 relationships. Figure 2 below shows the Symbols that may appear in an ER Diagram.

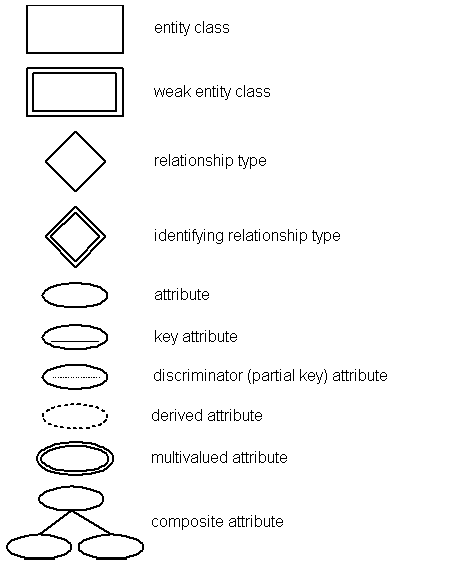


Figure . ER diagram Shapes

## ER Diagram Components

The following section describes ER diagram components that are supported by the ER Mapper program along with the properties each component contains. Each of these components map to the ER Diagram symbols from figure 3.

### Entity objects

An object that is distinguishable from other objects in the mini-world that must be represented in the database. In a relational database, an entity will be a relation. Entities can have different types. *Regular Entities* have a key attribute, and are *Strong Entities*. *Weak entities* cannot be distinguished by itself 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 3. Weak Entity Example* shows a weak Relationship where Teacher is the weak entity, and class is the Strong entity. The primary key of teacher is tId, code, where code references the primary key of class.

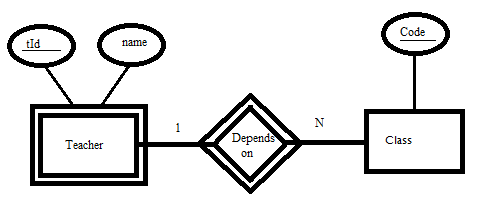


Figure Weak Entity Example

### 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 can be depicted by its cardinality which can be 1:1, 1:N or M:N. *Figure 4 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 3, Weak Entity Example,* the relationship Depends on represents an identifying relationship.

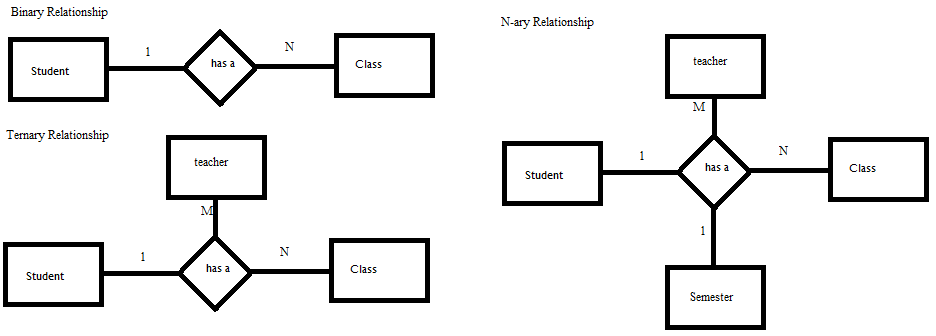


Figure Degree of Relationships

### Attribute

An attribute property describes an Entity object, and will appear as a column in a relational table. An attribute property may also describe a property of a relationship. In a relation multiple each attribute is considered to be a *candidate key.* From the candidate key, a *primary key* can be chosen as a unique identifier. A *super key* is a set of attributes in a relation schema, where no two tuples in will have t1[S] = t2[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 a*n attribute that is composed of several components

## Converting Relationships to Binary

Before normalizing the diagram, The ER diagram must be mapped to a relation schema. In general ternary and n-ary relationship can be represented as binary relations, which makes it easier to find functional dependencies. The following steps explain how to convert relationships to binary relations.

1. For each 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.

## Functional Dependencies

To create a relational database, the ER Diagram must undergo mapping to a schema. An important part of relational database design is functional dependencies which outline constraints between sets of attributes in the database. 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 t1[x] = t2[x], they must also have t1[Y] = t2[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. Once a relational schema has been defined the normalization process analyzes the schema based on its functional dependences placing it in normalized form. The benefit of placing your relational database into a normal form is that it decomposes relations which minimizes redundancy and prevents anomalies.

A good database design will be in a minimum of *Third Normal Form* as well as include *lossless join property* and *dependency preservation property*. 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 the purpose of this project, third normal form is sufficient. The *First Normal form* removes composite/multivalued attributes along with nested relations. In *Second Normal* *Form* requires that every non-prime attribute in a relation is fully functionally dependent on the primary key. Finally the *Third Normal Form* requires that no nonprime attribute is transitive, meaning each FD X->Z and Z-> Y. The *lossless join property* ensures that any instance of the original relation can be decomposed into smaller relations with no loss of information. And the *dependency preservation property* ensures that after the relation is decomposed each *functional dependency* still holds.

# Use Case Diagrams

The following section describes the ER Mappers use cases which describes the behavior and functionality of the system. *Figure 6. High Level ER Mapper Use* Case is a high level use case which describes the behavior of the ER Mapper system. It shows that they system only has one actor who has an option to create a diagram or normalize a diagram.

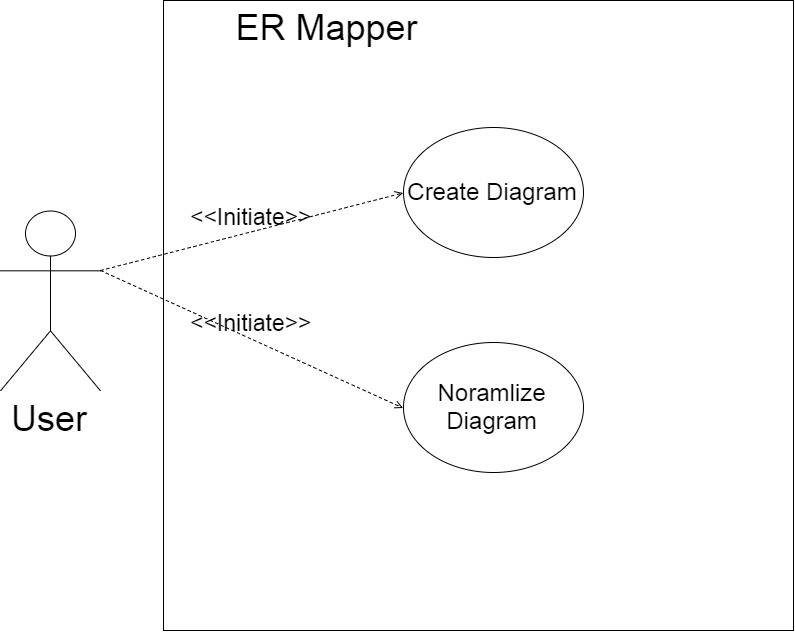
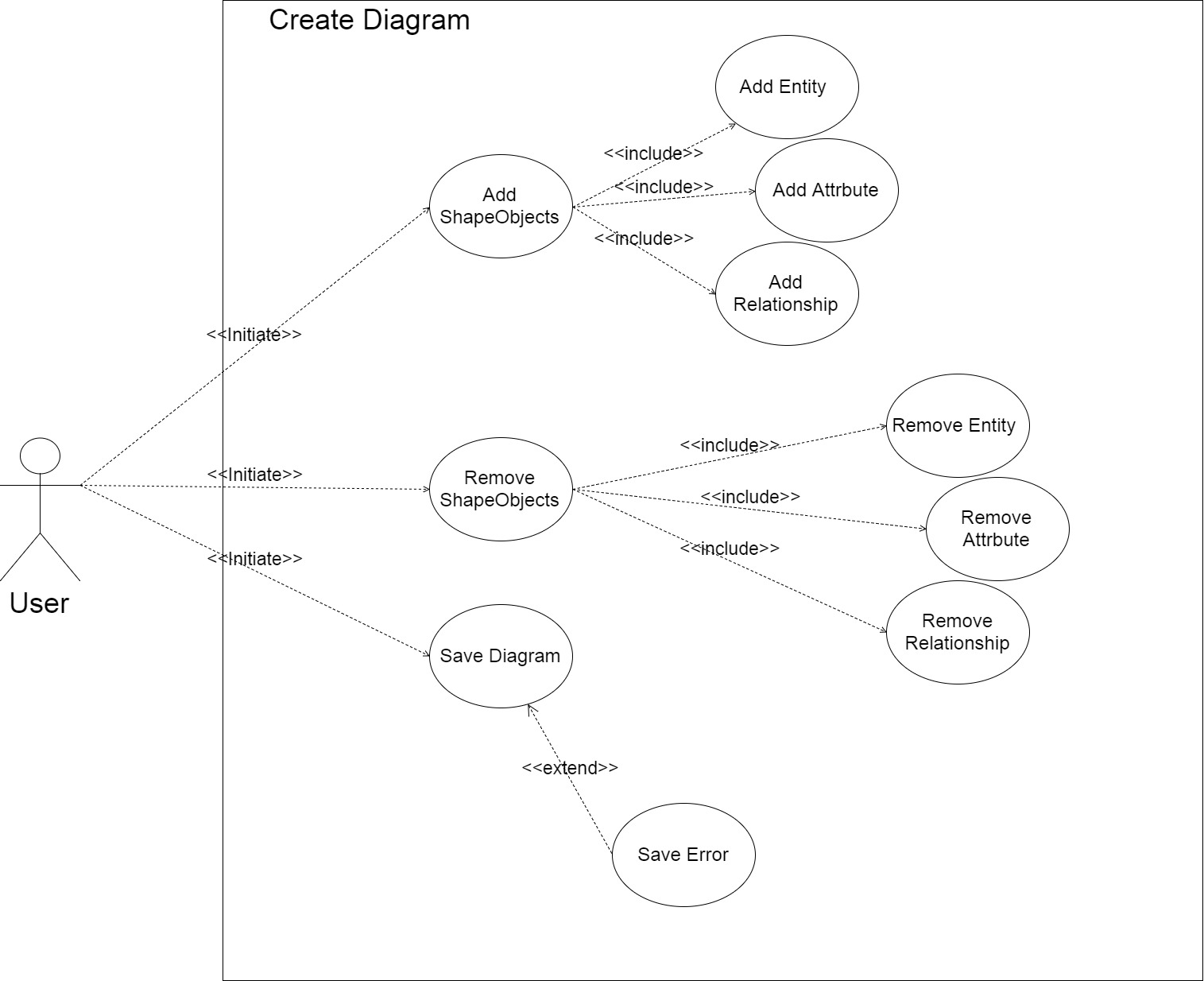


Figure High Level ER Mapper Use Case

The create diagram use case describes the functionality the system has to create an draw a diagram, as shown in *Figure 6. Create Diagram Use Case.* The system provides a blank canvas that the user can draw a diagram to, which allows them to create and delete Attribute, Entity and Relationship objects as well as save the diagram in an xml format.



The program also allows for a diagram to be normalize as shown in *Figure 7. Normalize Diagram Use* *Case*. In this process it takes the diagram and maps it to relations to create a Relation Schema, 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.

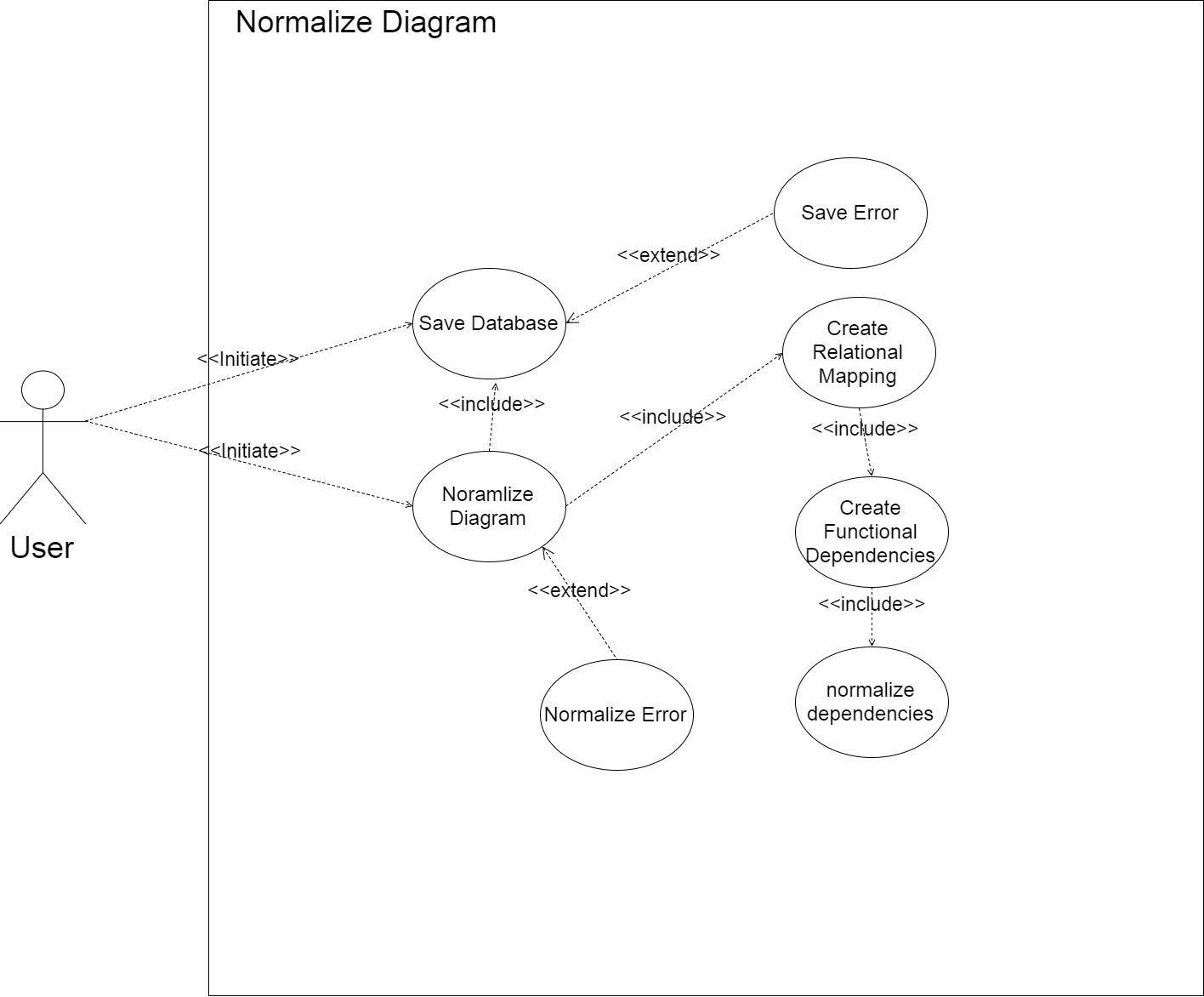


Figure . Normalize Use Case

## Use Case Descriptions

The below table describes the control flow of each use case and offers traceability back to its functional dependency.

Table Use Case Descriptions

|  |  |
| --- | --- |
| *Use Case Identifier* |  |
| *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 nprmalize the diagram **include Normalize Diagram** |
| *Entry conditions* | User selects Create new diagram |
| *Exit conditions* | User creates a diagram or exits the system |
| Traceability | FD |
| *Use Case Identifier* |  |
| *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 select to remove an object  2. the user clicks an object  a. the system removes the object corresponding to the object clicked. **Include removeEnitty, removeAttribute, removeRelationship.**  Or the user selects to save the diagram   * 1. 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. |
| *Use Case Identifier* |  |
| *Name* | **Normalize Diagram** |
| *Participating actors* | User |
| *Flow of events* | 1.if the user selects to normalize the diagram  a. The system create a relation schema. **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. |
| *Use Case Identifier* |  |
| *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. |
| *Use Case Identifier* |  |
| *Name* | **removeEntity** |
| *Participating actors* | User |
| *Flow of events* | 1. The user selects delete and clicks on an entity object    1. The system searches the list of drawn objects for the selected entity   If the entity is not part of a relationsip   * The sub objects are added to the shapebject list * The entity is removed from the shape object list   If the entity is part of a binary relationship   * The system adds the subobjects of the selected entity to the drawsnshapes 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   * The entity sub objects are added to the shapeobject list. * The entity is removed from the relationship   1. The system redraws the diagram. |
| *Entry conditions* | User selects delete an entity |
| *Exit conditions* | Object deleted and diagram is redrawn. |
| *Use Case Identifier* |  |
| *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 |
| *Use Case Identifier* |  |
| *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 |
| *Use Case Identifier* |  |
| *Name* | **addRelationship** |
| *Participating actors* | User |
| *Flow of events* | 1. User selects relationship button    1. The system creates a new relationship object 2. The user clicks one object    1. 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    1. The system sets the center of object as the second coordinates of the line.   If the relationship is between an entity and an attribubte   * 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   * The system checks adds the second attribute to the attribute list of the first attribute.   If the relationship is between two entities   * The system adds both entities to the relationship object. * The system creates cardinalities for each entity * The relationship is added to the shapeobjects list. * Each entity is removed from the shapeobjects list.   If the relationship is between an entity and an existing relationship   * The entity is added to the relationship object list * The entity is removed from the shapeobjects list * The entity gets a cardinality object   If the relationship is between an attribute and a relationship   * The system adds the attribute to the relationship   1. The system redraws the diagram. |
| *Entry conditions* | User selects create a relationship |
| *Exit conditions* | The diagram is redrawn. |
| *Use Case Identifier* |  |
| *Name* | **removeRelationship** |
| *Participating actors* | User |
| *Flow of events* | 1. User delete and clicks a relationship object    1. The system adds all sub objects of the relationship to the shapeobjects list.    2. The system removes the relationship from the shapeobject list |
| *Entry conditions* | Relationship is removed, and diagram is redrawn |
| *Exit conditions* | The diagram is redrawn. |
| *Use Case Identifier* |  |
| *Name* | **SaveDiagram** |
| *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 the a success or error notification |
| *Entry conditions* | The user selects save |
| *Exit conditions* | The user presses ok on the notification. |
| *Use Case Identifier* |  |
| *Name* | **Create Relational Mapping** |
| *Participating actors* | User |
| *Flow of events* | * 1. The system takes relationships in the diagram and applies the ER-> relational mapping rules to create all entities   2. The system creates a relation for each entity   If the entity is not weak   * adds the primary attributes of the entity to the primary key, and all attributes to the candidate keys.   If the entity is weak   * Adds the primary key of the strong entity to the weak entity primary key. Adds all attributes to the candidate keys.   1. If the entity has a multivalued attribute * The system creates a new relation with the attribute containing the values list as the primary key * All attributes in the values list are added to the candidate keys list   1. The system removes any temporary attributes   2. The system normalizes the schema. **Include create functional dependencies** |
| *Entry conditions* | The system creates a new relational schema |
| *Exit conditions* | The relational schema is created |
| *Use Case Identifier* |  |
| *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, else it is not.  d. the system performs normalization on the dependency set **include Noramlize dependencies** |
| *Entry conditions* | The system created the relational schema |
| *Exit conditions* | All functional dependencies are created |
| *Use Case Identifier* |  |
| *Name* | **Normalize dependencies** |
| *Participating actors* | User |
| *Flow of events* | * 1. The system finds the minimal cover of the dependency set   2. The system finds all candidate keys of the table   3. The system places any attributes not in the set in a table of their own. * If none of the tables are created contain a candidate key, then a new table is created with the candidate key   1. The system removes any redundant tables |
| *Entry conditions* | The system creates a dependency set |
| *Exit conditions* | The dependency set is normalized |
| *Use Case Identifier* |  |
| *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 |
| *Use Case Identifier* |  |
| *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 dispalys a prompt  **extends Save Diagram, Save Database** |
| *Entry conditions* | A save error occurs |
| *Exit conditions* | The user accepts prompt notification |
| *Use Case Identifier* |  |
| *Name* | **Normalize error** |
| *Participating actors* | User |
| *Flow of events* | * 1. the system gets an exception when trying to parse the ER Diagram, **extends normalize diagram**   2. the system displays an error prompt |
| *Entry conditions* | A normalization error occurs |
| *Exit conditions* | The user accepts prompt notification |

# Classes

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 ERdiagram or a Relational Schema. The logic package pertains to all classes that control the behavior of the components, including the activity classes.

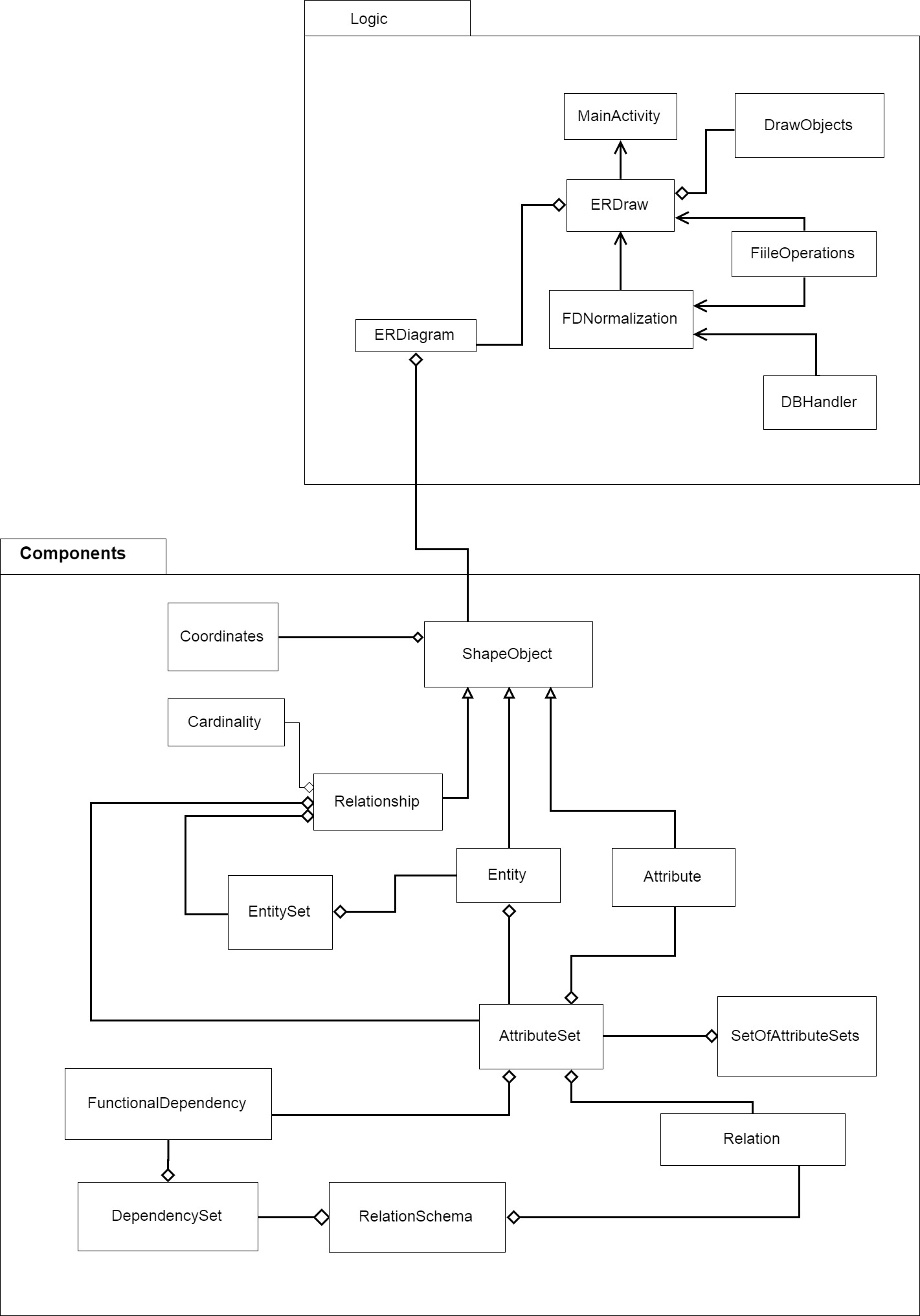


Figure . HighLevel UML Models

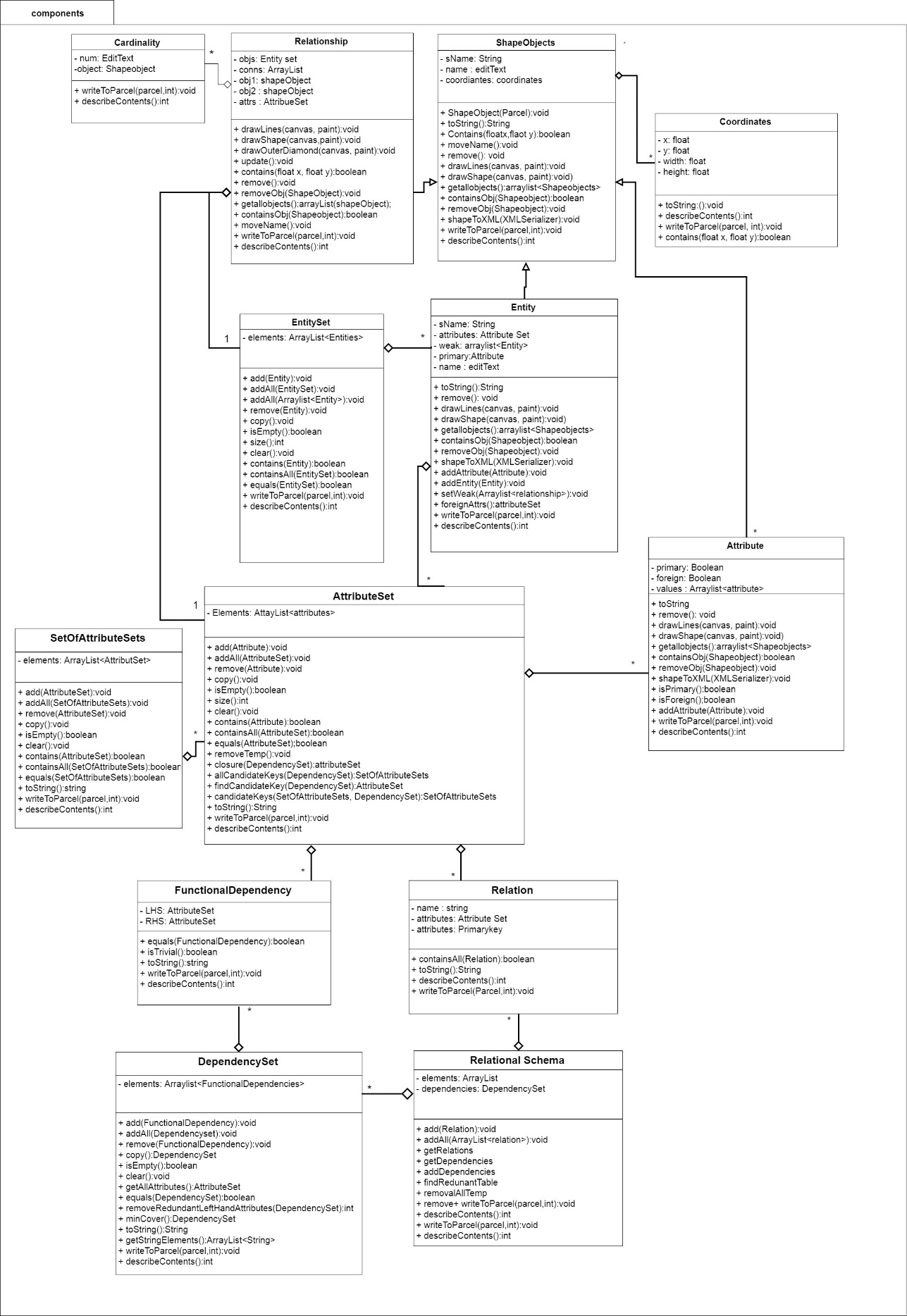


Figure . Compoenents UML Model

## ShapeObjects

An ER Diagram is built up from different ShapeObjects, including entities, attributes and relationships. ShapeObjects is an abstract class that contains each ShapeObject as a sub class. This allows for a single interface to be used for each object. Within this class there are several abstract methods that get overwritten by the sub classes defining specific behavior. Each of these objects contains a name, edit text and coordinates which allow form them all to be drawn to the canvas. The edit text makes it possible to change the name of any object because it has an event listener that will be called whenever the user changes its text field. Shapeobjects extend Parceable so that they can be passed between android activities. Parceable works, by decomposing an object into a parceable object using wrtiteToParcel(), and it then readsFromParcel() and calls a creator class to reconstruct the object from the parcel.

## Entity/Entity sets

As explained in the research section, an Entity represented a relational table, which models some aspect of the real world. An entity contains a list of attributes along with a list of weak entities. An entity is represented as a square, and contains a solid line to connect its attributes. For an entity to contain any weak entities it must be part of a relationship with a weak entity. In which case, the weak entity is drawn as 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. An entity contains its weak entities in an arraylist instead of an entity set because, when a parceable creates an Entity Set it calls the creator for an Entity, which causes the creator to get stuck in a look.

## Attribute/Attribute sets/SetOfAttributeSets

An *attribute* can be a property of an entity or of a relationship. An attribute is represented by an oval. Primary attributes are used to uniquely identify an entity. In order to create an primary attribute it can be double tapped on the canvas. The attribute is then set to Primary and its name gets an underline. In the case of a weak entity a foreign key is required which gets represented with a red underline. Along with being primary and foreign, an attribute can be multivalued. i.e. it can be broken down into separate components, such as a date of birth 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. Similar to an entity when a parceable creates a attribute set it calls the attribute creator and gets stuck in a loop, which is why all multivalued attributes are stored in an array list instead of an attribute set. 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. Therefore when any attribute is created it is called “object” + count, where the count is incremented for every attribute created making it unique for every attribute. The *SetOfAttributeSets* class holds a unique set of *attribute sets*.

## Relationship/Cardinality

A relationship represents a connection between two or more entity objects. They are represented by a solid line and a diamond. If the relationship is an identifying relationship, it gets a diamond with a diamond outline, and has two solid lines to show total participation. This class contains an attribute set, representing any properties of the relationship. Along with an entity set, that 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 to model 1:1, 1:N and M:N relationships.

## Relations/Relation Schema

The relation class represents an relational table, derived from an entity object and its attributes and relationships. A relation contains an attribute set of attributes, 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 Relation Schema, represents a unique set of relations that make of the database relation schema. When creating a Relation schema, 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
   1. Choose one of the relations-say S-and include a foreign key in S the primary key of T.
   2. 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
   1. the relation S that represent the participating entity type at the N-side of the relationship type.
   2. Include as foreign key in S the primary key of the relation T that represents the 1 side of the relationship type
   3. Include any simple attributes of the 1:N relation type as attributes of S.
3. For all Binary M:N
   1. For each regular binary M:N relationship type R, create a new relation S to represent R.
   2. 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
   3. 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
   1. For each multivalued attribute A, create a new relation R.
   2. 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.
   3. 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.

## 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, and a right-hand side with all nonprimary attributes of a relation. A dependency set contains a list of unique functional dependencies.

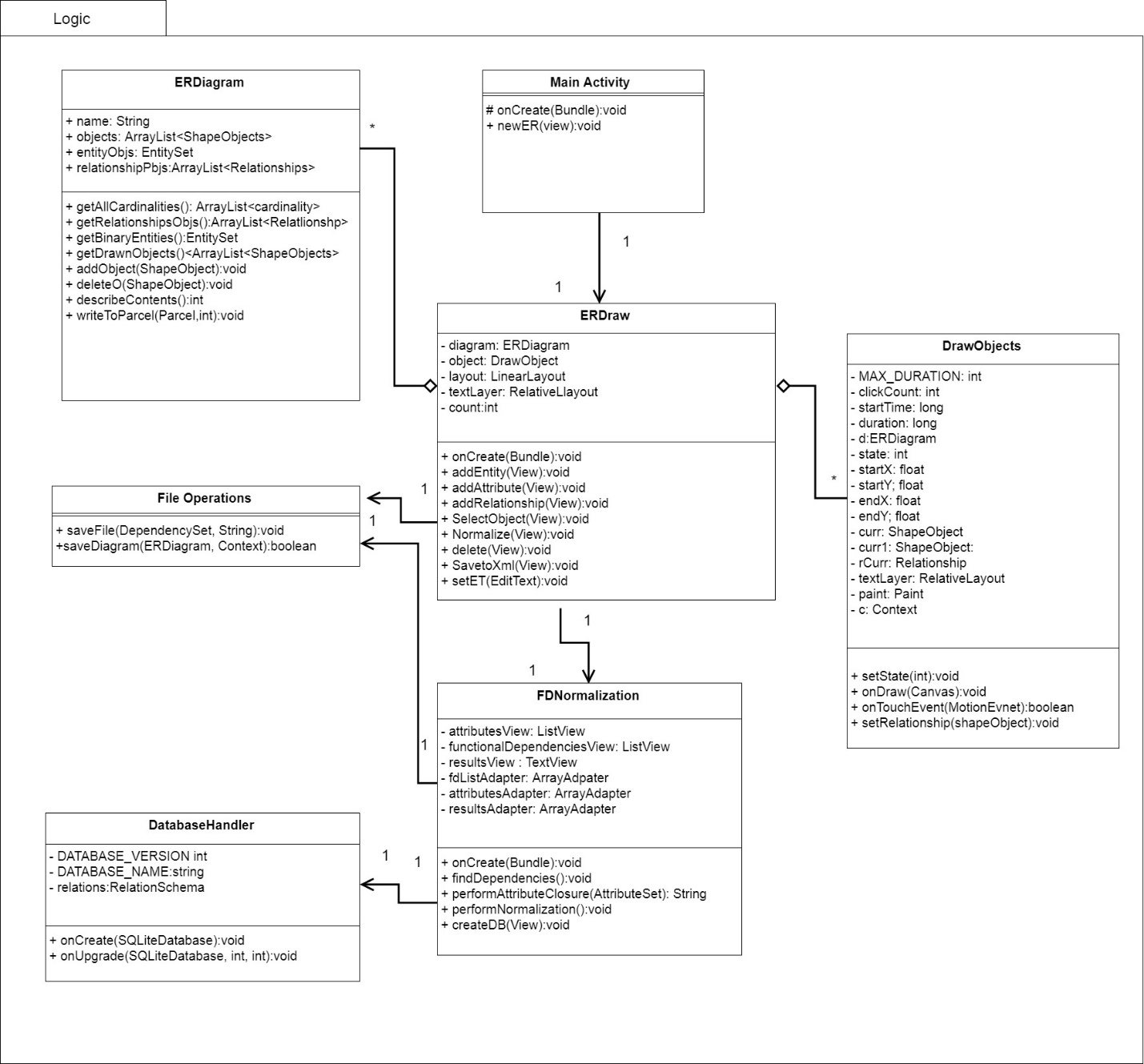


Figure . Logic UML Model

## Activity Classes

As part of an android app an activity represents a single screen with a user interface (Android.com). *MainActivity*, *ERDraw* and *FDNormalizations* are each activity classes. 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 user may also select the normalize button which launches the FDNormalization activity, which creates a new interface that displays the relational schemas details.

## 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 list of shapeobjects. As each object forms a relationship attributes and entities are added to their corresponding owner, and they are removed from the general shapeobejct list. In doing this it ensures if there exists 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 list. Now say 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 shapobject list and the relationship is added. To draw this relationship, the relationship drawLines() method is call to draw the lines between the entity and relationship, then the draw shapes method is call to draw the actual entity and attribute shapes. There is a function called getBinaryEntities(). This function 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 converts it to a binary relationship. 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 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** to “-1”, which allows for the entity to be considered valid while it gets mapped to a relation. Once getBinaryEntities 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 Relations / Relation schemas. Once the relation schema is created, there is a function removeTemps() that removes the temporary attributes from the schema.

## Draw Objects

The ERMapper starts by creating a blank canvas that allows users to draw an ER diagram. In the drawing phase users can create Entities, Attributes and Relationships between each. Users may select an option to create an object, delete objects, save the Diagram or normalize the diagram, as shown in red rectangle Figure 1 below.

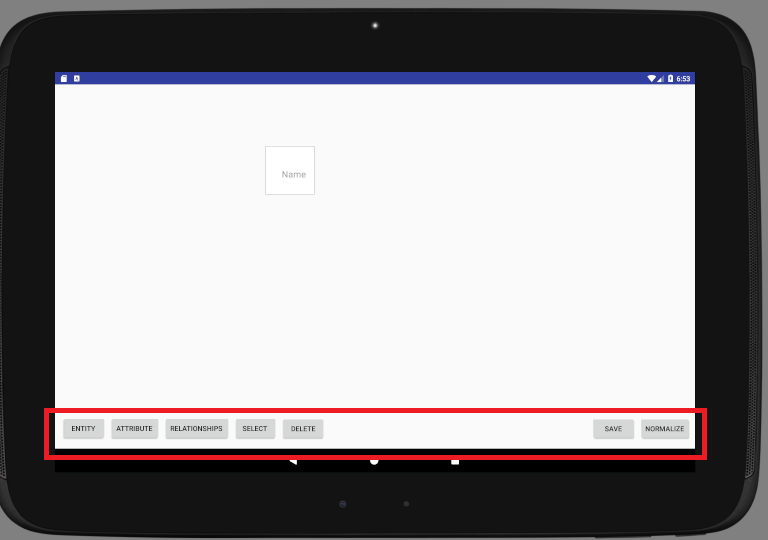


Figure ERMapper Drawing

The program contains a class called ER Diagram which stores an arraylist of shape objects 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 1 shows the canvas of the page. The user can select to create an entity, attribute or relationship. Upon doing so the object is create along with an edit text object to store its name and its set of coordinates and it is added to the object list. Unless the object is a relationship, in which case 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 search its list of objects and draws the correct shape based on the object type onto the screen at its correct coordinates. The canvas also has a motion event listener which activates when the user clicks the screen. It checks if the user has click 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 the they can click normalize to normalize the objects and create a relation diagram. *Figure 6* below is an image of a complete ER Diagram.

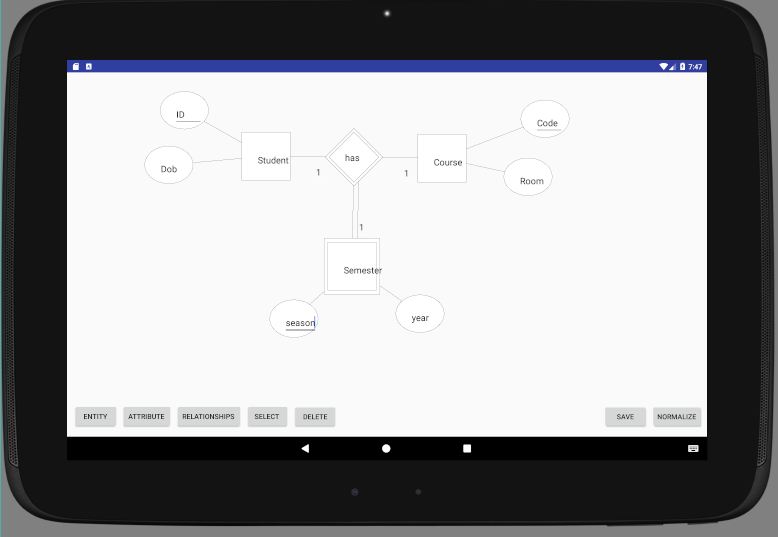


Figure Completed ERDiagram

## FD Normalization

To normalize the diagram, one must press the Normalize button from the diagram. Upon which 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 to the system to the screen and offer an object for the user to create a database. *Figrue 12. Normalized ERDiagram,* is an example of the FDNormalization activity. In the results section the following information is displayed:

1. All attributes
2. All functional Dependencies
3. Minimal cover
4. Minimal cover with left had sides merged
5. Checks that the minimal cover is equivilant to all original functional dependencies
6. All attribute keys of a table
7. Lossless-join and Depenency Preserving, 3NF tables
   1. This are the tables that will be used to create the database.



Figure Normalized ERDiagram

In order to get all the attributes, functional dependencies and results, the class takes the created relational schema, normalizes it and performs attribute closures. The FDNormalization class calls the getBinaryEntities() from the erDiagram class which allows each entity goes through ER to Relation mapping to be decomposed into a relational schema using the from the book *Fundamentals of Database Systems* by Ramex Elmasri & Shamkant B. Navathe.

1. For all Regular entity types, assign a Relation, pick a primary key
   1. if the primary key is a complex attribute: all attributes will be included
   2. 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 all entities are decomposed into binary relations, a new relationSchema is created were ER to relationship mapping rules (discussed in the section on relation schemas) are used to map the entities to relations. This prevents duplication of data and ensures that each relationship is converted to its proper entity. Because every entity object is stored as a shape object for drawing purposes, not all entities would yet be identified by the system, and the end result would be missing those Relations.

Once the relation schema is mapped each functional dependencies is identified where for any primary key in a relation must functionally determine all other attributes in the relation. To do this the program looks sets the primary key to the left hand side of the functional dependency and all other attributes to the right hand side. While it is doing this it ensures that there are no duplicate attributes.To ensure that the ER Diagram has a good database design, it will go through the normalization process to *Third Normal Form* keeping the *lossless join property* and *dependency preservation property.*

### Normalization

To convert relations into its normal form one can follow the steps provided in table 14.1 from *Fundamentals of Database Systems.* The code provided by Professor Lou Nel, handles the normalization contains the code to maintain *lossless join property* and *dependency preservation property.*

Table 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). |

*Figure 7, Normalized ER Diagram,* shows the screen that gets displayed after the normalization process which displays all attributes, functional dependencies, and in the Results lab a screen out put that contains 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.

The algorithm to Create Lossless-Join, Dependency Preserving 3NF tables, was provided by Lou Nel and based on the 4 step algorithm 16.6 presented in Elmasri and Navathe 6th ed. Which decomposes a set of attributes (universal relation) 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 Fmwith 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 theattributes, then create a table consisting of a candidate key
4. Remove redundant tables. If any table is a projection of another (has all its columnsappearing in another tables, then remove that table from the decomposition

### Attribute Closure

The attribute closure algorithm was provided by Lou Nel and

# Results

The ERMapper program successfully allows for users to create basic ER diagrams on an android app and automatically generate a relational database. Some issues with the system include that you order matters when creating a complex attribute. Foreign keys are not represented with a dashed line. Along with this sometimes double clicking is ends up moving an object. The system also does not have a clear understanding of what the user wants as the ER diagram is ambiguous. Therefore, if you make an error in your diagram the program will not be able to fix it.

# References