Entity Relationship Diagram Mapping

by

Cheryl Dunn

100963953

Prof. Louis D. Nel

School of Computer Science

Carleton University

Ottawa, Ontario

# 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 database. 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. For 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.

# Thesis

ER Diagrams are used in the database development process, creating 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. Throughout this project, I will walkthrough 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.

The following are instructions on how to setup the program to run on an 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.

The following are instructions on how to setup the program to run on android device.

1. 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 1 Green Run Arrow

# Functional Requirements

The following are the requirements that must be met by the ERMapper program.

1. User must be able to create/delete *entities*
   1. Entities require a primary key, which is a unique name
   2. In the ER diagram the Entity will be represented by a square
2. Users must be able to create/remove attributes
   1. *Attributes* must belong to an *entity*
   2. *Attributes* can be composite or multivalued
   3. In the ER diagram the *attribute* will be represented by an oval
   4. The *Primary* *attribute* will be the Primary key of the *entity*, depicted with an underline
3. Users must be able to create/ remove *relationships*
   1. Connections can be between2 *entities*
   2. Connections can be between an *entity and an attribute*
   3. Connections between 2 *attributes* show composite or multivalued *attributes*
   4. Connects will be depicted with a Line
   5. *Relationships* will be depicted with a diamond
4. Users must be able to select a *attribute, entity or relationships* and move it around the screen
5. The system must create an *ER Diagram* that contains a set of Entities with its attributes and connections
6. The system must convert the ER diagram into a relation schema
7. The system must identify all functional Dependencies
8. The system must normalize the relations into Third Normal Form maintain lossless join property and dependency preservation property.
9. 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.

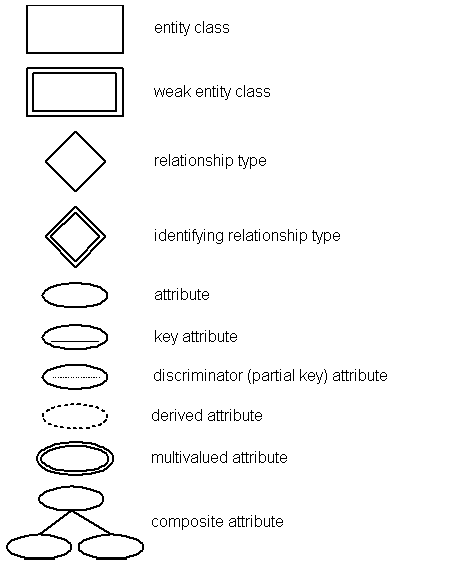


Figure 2. ER diagram Shapes

## 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.

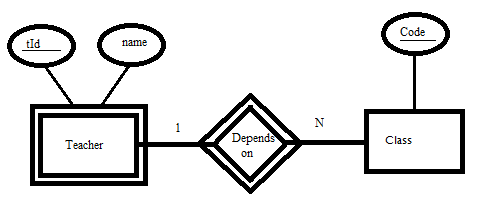


Figure 3 Weak Entity Example

### 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.

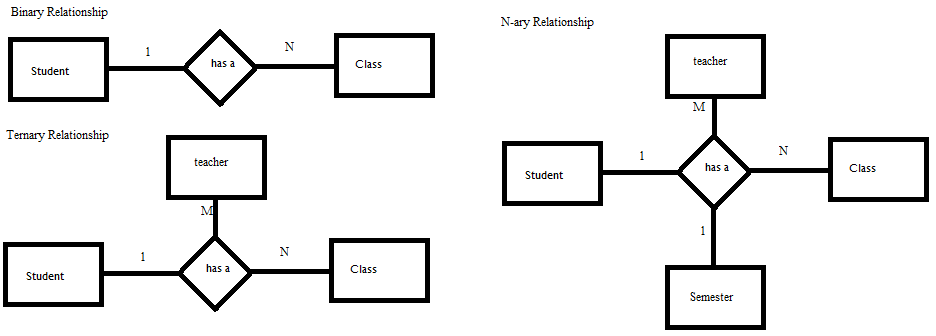


Figure 4 Degree of Relationships

### 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.

# ERMAPPER

## Class Diagrams

The following are class diagrams that show how each class in the program are related, along with their attributes and methods.

### ER Draw

The ER draw class is an activity class that calls the draw object class and handles the functionality of pressing a button to draw. When a button is clicked, this class will check what object you want to create and add it to the shapobject list in your ERdiagram, adds a editText view to the entity and attribute objects so that its name can be edited, and then draws the object to the canvas using a drawObject object. Upon initialization every object is drawn in the upper left of the screen, and the object name is null. Every editText view gets an onFocusChangeListener, which updates the name of the object whenever the user edits it. For a relationship, since we do not know if it will need a name or not, the edit text is not initialized in this method. As we also do not know the coordinates of the relationship, when the relationship is initialized, the object is passed to the DrawObject class so the user can draw the line between anyobjects.

### 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 ensure 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. Once the user is finished drawing their diagram and they decided to normalize the diagram, a Parceable diagram is created and passes on all relationship objects to the fdnormalization class. From the relationship objects, we can obtain all information need to create relations and functional dependencies.

## Drawing the Diagram

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, as shown in red rectangle Figure 1 below.

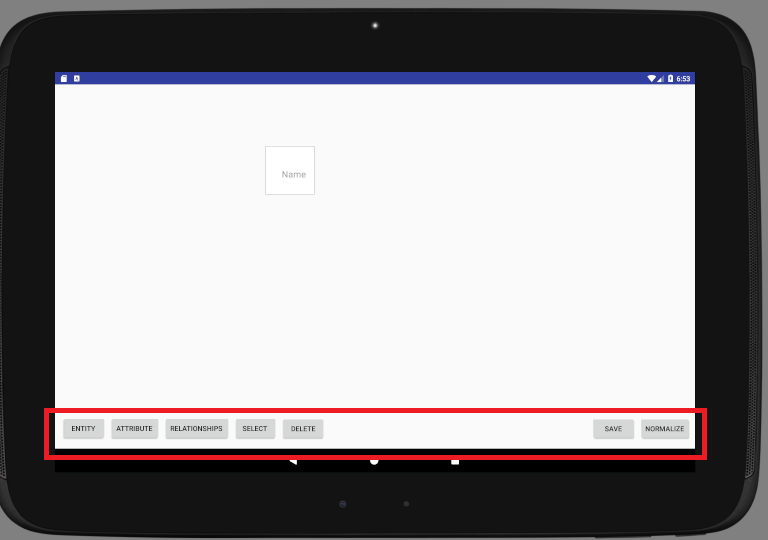


Figure 5 ERMapper Drawing

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.

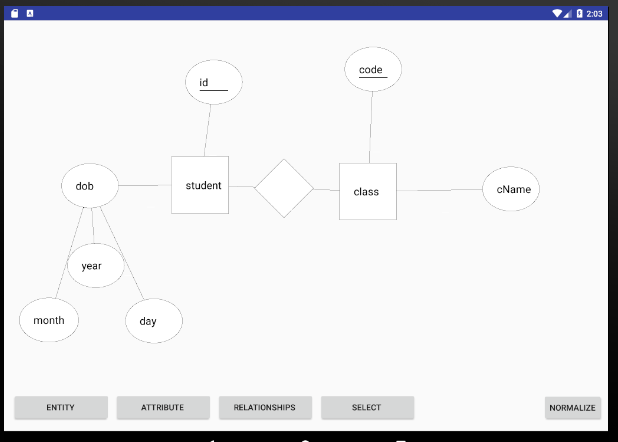


Figure 6 Completed ERDiagram

## Normalizing the Drawing

To normalize the diagram, one must press the Normalize button from the diagram. Upon which the diagram 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.

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

In the FD Normalization Class, steps 3-6 are performed first on the entity objects before they get converted 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.

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.

In the Program, 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, as per steps 3 to 6. After this process all entity objects and attributes have been identified. Next steps 1 and 2 are performed where for each of the identified entity, a relation is created and it looks at all attributes, and if the attribute is primary or foreign it belongs to the primary key, else it is a regular attribute of the relation.

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 a 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 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). |

*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.



Figure 7 Normalized ERDiagram

# Work to do

Next to be done in the system is to complete steps 3,4,5 and 7 of the normalization process which includes handling more complex relationships. To do so I need to create relationships that show cardinality. I will also have to create the connection to a sql database take the relations and create the database. Testing and other debugging will also be required. Other things that I would like to include if time permits is adding a way to remove objects so that you do not have to restart, refactoring the code to simply and remove any duplication as well as add more error checking.

# Known issues

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.

# Work Schedule

The table below is an updated version of expected time to finish the project.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |  |