Project Design Report

Fossils of the Southeast US

Front Row

3/29/2019

Jeffrey Bruggeman

Kevin Seveur

Julio Soldevilla

Philipos Zegeye

# Introduction

The database that Front Row is designing is for the representative data of fossils collected and identified by Mike Bruggeman over the last 55 years. While many of the fossils are from all over the globe, most of the collection is from hundreds of sites around the Eastern and Southeastern United States. The data was originally compiled to assist Mike in printing high quality labels for his reference fossil collection, however now he would now like to create a database to assist people in quickly identifying fossils they find all over the world. Along with 60 attributes for describing and grouping these fossils and their identifying information and around 10,000 tuples, there is also several thousand images that correlate to many of the specimens. Some of the important information to keep in mind when reviewing this database is that the client has established his own reference and “taxa codes” for each fossil that are unique to his own collection. For much of his identification he used books that were published on the specific area he was collecting in and he often recorded the identifying author, as well as the Dewey Decimal Classification of the book. However, as his textbook collection reached an outstanding size, he began his own book numbering system that he often recorded in a different attribute. These bits of trivia are indicative of the scale of this data collection.

The main motivation behind this project selection was that Mike’s son Jeffrey has helped him collect these fossils for the last 25 years and would like to assist in the creation of a tool to help others that share this hobby. After Jeffrey presented the idea to the group, we all agreed that this idea was interesting, but also that the state of the data posed a unique challenge that might accurately represent the work we would all do in the real world. The data was created over 10 years by a single person with no consideration of using a single style or that anyone else would even ever see it, so there are many interesting challenges to creating a database here.

The main usage of this database will eventually be through a GUI applet that will display a grid of fossils images along with their identifying information based on many different selected search criteria. Search criteria can include anything such as age, state, formation, any taxa, Also, there will be a screen that can display two fossils at a time for quick comparison, as well as detailed information about them. Further requirements posed by the client potentially would go well beyond the scope of this class, but these goals will be pursued by Jeffrey after this class has finished. While this project may not include a GUI upon the completion of this class, the queries and structure of the database will be based off of the criteria established by the client.

# Requirements Analysis

## Data Requirements

This database will be used to store fossil information that is mostly made up of character strings. Because the existing data was hand entered and constantly updated, many nulls currently exist in the data but will be filled in with “null” or “0” depending on the value required to maintain ordinality. The entity Fossil is used to describe the fossil itself such as it’s binomial enumeration of genus and species, any pictures of it that exists, the location number from where it was found, an attribute called “ImageID” that each fossil is associated with, and other such identifying information. The Reference entity is descriptive of the texts or websites that fossils were identified through and includes as much information about the text that seemed relevant to the ID, such as page number the fossil is on, the author(s) name and publication date. The Location entity provides information about the locations the fossils were discovered, such as area, age of the fossils from that area, and text paths leading to any map images of the area. The taxa entities: Family, Order, Class, and Phylum maintain the various taxa hierarchy that each fossil belongs to. Finally, the two author multivariate attributes exist to maintain the authors of a reference text and the ‘author’ or authors of a fossils etymology. ImageID is a generated number that is unique to each unique fossil. All the taxonomies, such as species, genus, family, class, order, and phylum are string values. Author is a string, when surrounded by parenthesis it signifies that the fossil species was later moved to a different genus. RefNo is an integer assigned to journals and books, currently it is in the range 1-853 but could be increased by adding more books. LocNo is the region code, a hyphen, then an incremental number for each new locality in that region. Taxacode is an alphanumeric combination of the client’s creation where the first number is the phylum number, the next letter is the class letter, and the next number is the Family number. All of these are unique to the client’s data and don’t exist anywhere else in the field. PH\_Numbers is an alphanumeric string that refers to the image name. XC and RX are 1 or 0 to indicate true or false. Size is an integer measurement in mm. Price and RefPrice are dollar amounts. Special, Uncertain Loc, Specific Loc, Visited Loc are all True or False. Dewey is a float value with up to 3 decimal values, or NA (should be changed to null) if record is not a book. Date is the year the text was published. Pages is a range of integers or a single integer. Finally, the following fields are just character strings of varying lengths: RefPgFig, txtImageName, Type, Common Name, Photo, Figure, Num Specimen, LName, Period, LocLabel, Country, County, Series, Stage, Group, Formation, Member, LMap, GMap, Memo, Level, OrgCode, Title, Subject, Keyword, Author\_Text, Publ, Source, RefMemo, RefType, ISBN, and GenName.

## Functional Requirements

The Functional Requirements for this database consists retrieving specific data and modification of data. This includes the fossil taxonomy, location found, period it was from, and associated reference materials. This information should be able to be retrieved separately or in any combination of attributes. For example, the user should be able to retrieve the genus or the genus and family. Other requirements include that the user should be able to get the image for any fossil that the user wants, including available location images or map images.

* Functional Requirement 1: The user must be able to get full taxonomic information for each unique fossil
* Functional Requirement 2: The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils found in a specific location
* Functional Requirement 3: The user needs to be able to view the larger image side by side with another larger image and some associated information (such as family, genus, species, location, formation, taxacode, locno) about both of them for identification comparison.
* Functional Requirement 4: The user needs to be able to print labels for any specific fossil or a group of fossils with information in the database- the labels have the following information: Genus, Species, Phylum, Class, Period, Formation, Location, Taxacode, and LocNo
* Functional Requirement 6: The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils that were identified through a specific refno
* Functional Requirement 7:The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils from the same formation
* Functional Requirement 8:The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils in the same Phylum, Class, Order, Family, or Genus
* Functional Requirement 9:The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils where the “Special” attribute is True to check which fossils are not currently in the physical collection but rather on display elsewhere.
* Functional Requirement 10: The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils from the same period

# Conceptual Design

## Entities and Attributes

### *Entity 1: FOSSIL*

### 

* + 1. Figure 1: Attributes and Relationships of Fossil

*Entity*: This entity is descriptive of each fossil that is in the collection, usually paired with an individual image that was taken.

*Attributes*

* ImageID: Unique element identifier, essentially SSN of each record. Foreign key to Set, Fossil Name
* LocNo: Foreign Key that links to LOCALITY. Location number client assigned each new location he went to. Has the form of region, then a hyphen, then a new incremental number.
* Taxacode: Composite code created from Phylum, Class, and Family. For example 15E-37 refers to 15 (the phylum number for Mollusca), E (the class letter for bivalves within the phylum mollusca), and 37 (the number assigned to the Family Semelidae).
* RC: Reference collection (1 or 0 for T/F)
* XC: Extra collection (1 or 0 for T/F)
* Special: (T/F) Indicates that the fossil is stored in a place other than collection
* PH\_Numbers: Composite number similar to Taxacode but with more specific identifying numbers and letters to attribute each fossil to an image in one of many folders.
* Common Name: Common fossil name
* txtImageName: plain text of fossil image location
* Photo: Smaller resolution image path for displaying multiple images together
* Type: Plain text descriptive of misc information about fossil, mostly null
* Fossil\_Author: Describes the founder of a fossil
* RefNo: Foreign Key that links to REFERENCES. Refers to the text that the fossil was identified by, RefNo is a incremental numbering of books that exist in clients’ own library.
* RefPGFig: Specific page and figure where the fossil is named within text.
* Size: Approximate size of fossil in mm
* Price: Price if client ever sells this fossil.
* Figure: Image from textbook where identified, if available
* NumSpecimen: Number of specimens in collection, if available
* GS-ID: A unique ID that identifies a combination of genus and species

*Relationships*

* References: Each RefNo refers to a book within the clients’ library, most elements have a RefNo, but not all.
* Locality: Every element in fossil has exactly 1 locality. Most localities have many fossils. LocNo refers to a specific fossil locality.
* Genus: Every fossil has exactly 1 genus species combination, every genus has at least 1 fossil.

*Primary Key*

* ImageID: an iterative number assigned to each new tuple. Because this entity is descriptive of a collection of physical fossils that exists within a collection, some specific species from specific locations are repeated for various reasons, so a unique identifier was generated upon creation of each tuple in data table.

### 

### *Entity 2: REFERENCES*

### 

### 

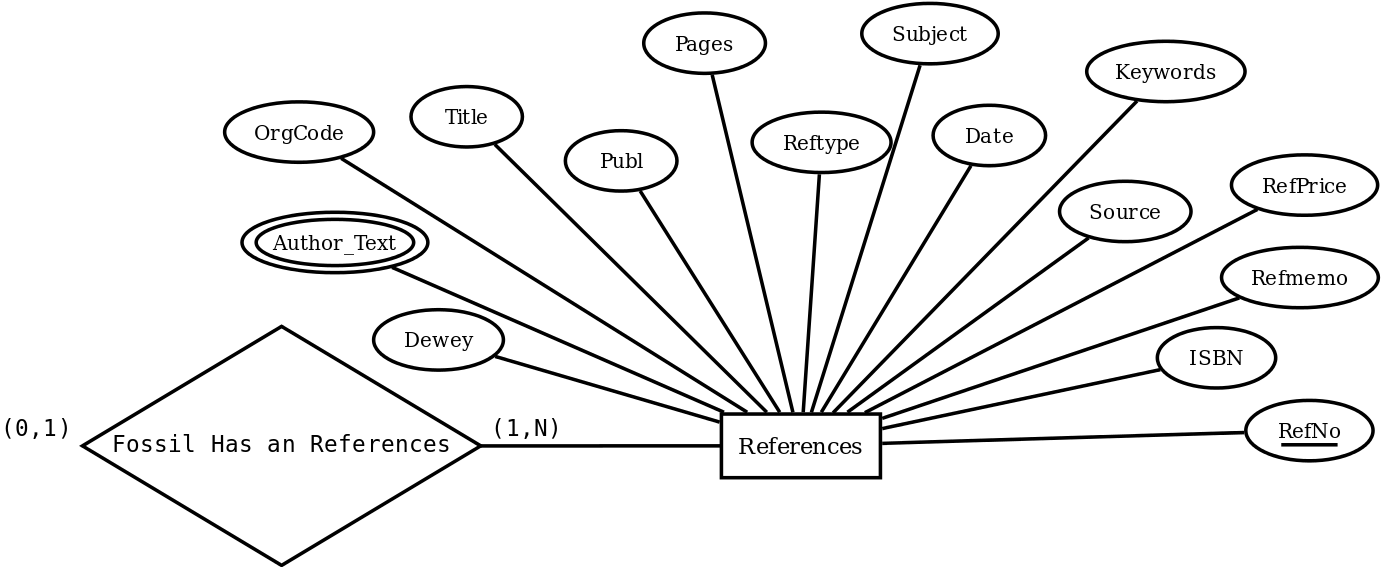


Figure 2: Attributes and Relationships of References

*Entity*: This entity is descriptive of the reference textbooks used to identify the fossils in the collection.

*Attributes*

* RefNo: Unique identifier for each element in REFERENCES entity.
* Dewey: Dewey Decimal number
* Orgcode: Publishers organization code.
* Title: Title of text
* Author\_Text: Author(s) of text
* Publ: Publisher of text
* Pages: Pages the fossil is identified on
* Reftype: Type of text: journal, book, etc
* Date: Date of publication
* Subject: Subject of text
* Keywords: Keywords to help find texts
* Source: Where the text came from
* Refprice: Price of text if available
* Refmemo: Note about text
* ISBN: ISBN of text

*Relationships*

* Fossils: Ordinality listed in fossils section.

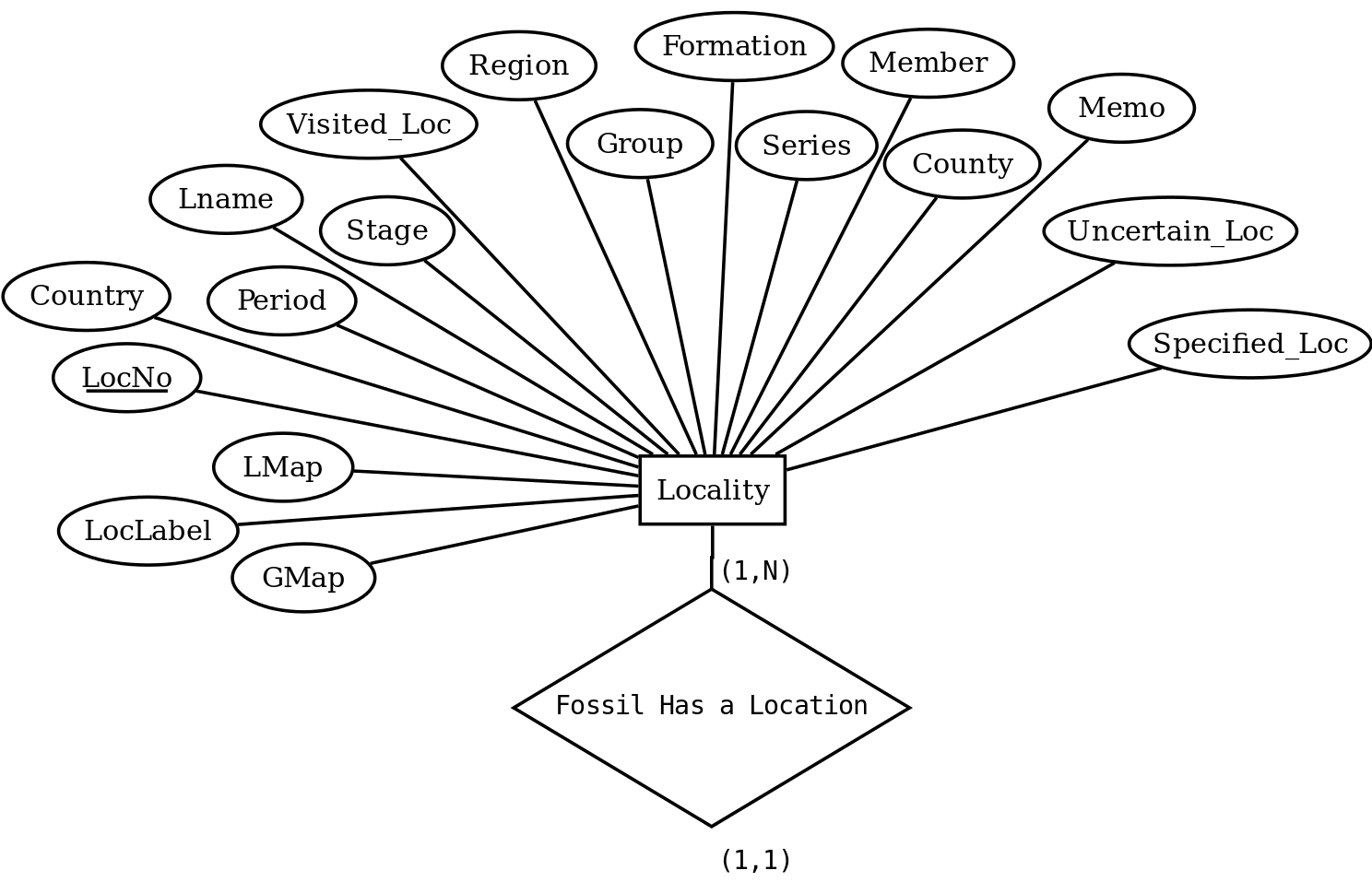
*Primary Key*

* RefNo is the unique identifier for each text listed in this entity.

### 

### *Entity 3: LOCALITY*

### 



* + 1. Figure 3: Attributes and Relationships of Locality

*Entity*: This entity is descriptive of each location fossils were found at or were known to be found if bought or traded for. Some locations are approximate because they have never been visited by the client.

*Attributes*

* LocNo: This is the primary key, as well as a foreign key that links to the LocalityPics entity. It is composed of a few numerical characters that refer to the state or country the fossil was found, then a hyphen, then an incremental number for each new location in order discovered by client.
* Region: Redundant, this is the characters before the hyphen in LocNo
* Country: Country of discovery
* LName: Common location name, usually a nickname.
* Period: Period within an era.
* Stage: Sublevel of series
* Group: Super level of formation
* Formation: Strata within an age
* Series: Sublevel of period
* Member: sublevel of formation
* County: County in US
* Memo: Note about location
* LMap: Map image file location
* GMap: Other map image file location
* LocLabel: Label for location
* Uncertain Loc: T/F for uncertain where it is located
* Specific Loc: T/F for specific location known
* Visited Loc: T/F for the client has visited this location

*Relationships*

* Fossils: Ordinality listed in fossils section.

*Primary Key*

* LocNo: Uniquely describes all attributes in locality

### *Entity 4: Family*

### 

### 

### 

* 1. Figure 5: Attributes and Relationships of Family

*Entity*: This entity is descriptive of the genus that a fossil belongs to

*Attributes*

* Order:The scientific name for the order associated with this family
* Family: Primary key, The scientific name of the family associated with this genus

*Relationships*

* Order: Family has a 1 to 1 relationship with Order
* Genus: Family has a 1 to N relationship with Genus

*Primary Key*

* Family: As the taxa above Genus, family will uniquely identify all genus under it

### *Entity 5: Order*

### 

### 

### 

* 1. Figure 6: Attributes and Relationships of Order

*Entity*: This entity is descriptive of the order taxa and how it relates family to class

*Attributes*

* Order: The scientific name for the order associated with this family
* Class: The scientific name for the class associated with this order

*Relationships*

* Family: Order has a 1 to N relationship with Family
* Class: Order has a 1 to 1 relationship with Class

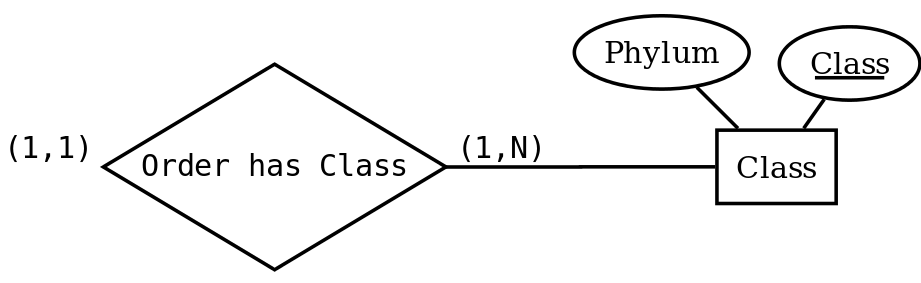
*Primary Key*

* Order: A class will uniquely define all order names within its taxonomy.

### *Entity 6: Class*

### 

### 



* + 1. Figure 7: Attributes and Relationships of Class

*Entity*: This entity is descriptive of the class taxa and how it relates order to phylum

*Attributes*

* Class: The scientific name for the class associated with this order
* Phylum: The scientific name for the phylum associated with this order

*Relationships*

* Order : Class has a 1 to N relationship with Order

*Primary Key*

* Class: Class uniquely identifies it’s orders

### *Entity 7: Genus*

### 

### 

### 

* + 1. Figure 8: Attributes and Relationships of Genus

*Entity*: This entity is descriptive of the genus taxa. This is the lowest taxa recorded in this database.

*Attributes*

* Genus: the genus part of the binomial nomenclature for all fossils
* Species: the species part of the binomial nomenclature for all fossils
* Family: The scientific name for the family
* GS-ID: A unique ID that identifies a combination of genus and species

*Relationships*

* Family : Class has a 1 to 1 relationship with Family
* Fossil : Class has a 1 to N relationship with Fossil

*Primary Key*

* GS-ID: GS-ID uniquely identifies it’s orders

## Relationships

## *Relationship 1: Fossil Has a Reference*

### 

*Relation*: Shows the relation between Fossil and References. It describes whether a fossil has a source of information such as a book or journal.

*Attributes*

* Foreign Key: RefNo relates each Fossil to a specific reference

*Cardinalities: Every fossil does not need to have a journal or book that references it, but at most can have one reference. Every Reference must reference a fossil and every reference can be used to reference multiple fossils.*

## *Relationship 2: Fossil Has a Classification*

### 

*Relation*: Shows the relation between Fossil and Genus entities. It describes whether a fossil has a a classification.

*Attributes*

* Foreign Key: GS-ID from Fossil relate a specific fossil to its Family

*Cardinalities: Every fossil must have a classification and at most can belong to one main classification. Every classification must describe a fossil, but certain classifications can apply to multiple fossils.*

## *Relationship 3: Fossil Has a Location*

### 

*Relation*: Shows the relation between Fossil and Locality entities. It describes if the location of the fossil origin is know.

*Attributes*

* Foreign Key: LocNo is unique to each specific specimen and so is the location. We can’t use genus and species here because sometimes you will find the same fossil at multiple locations.

*Cardinalities: Every fossil must have a location of origin and at most can only originate from one location. Every location must correspond to the origin of at least one fossil, but each location can be used as an origin for more than one fossil.*

## *Relationship 4: Family has an Order*

### 

*Relation*: Shows the relation between Family and Order entities.

*Attributes*

* Foreign Key: Family will contain a foreign key that will link to Order’s primary key.

*Cardinalities: A Family has a single Order, but an Order can have many Families.*

## *Relationship 5: Order Has Class*

### 

*Relation*: Shows the relation between Order and Class entities.

*Attributes*

* Foreign Key: Order will contain a foreign key that will link to Class’s primary key.

*Cardinalities: An Order has a single Class, but a Class can have many Orders.*

## *Relationship 6: Genus has a Family*

### 

*Relation*: Shows the relation between Genus and Family entities.

*Attributes*

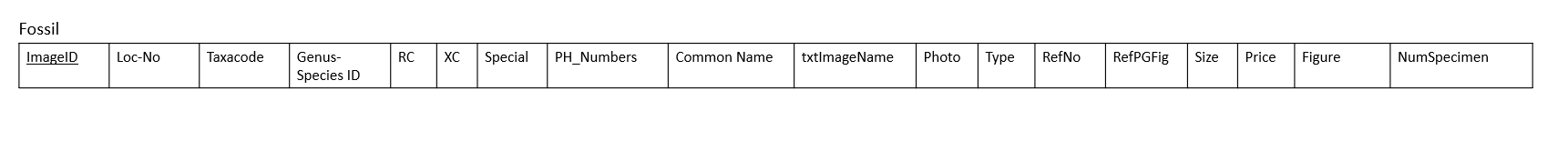
* Foreign Key: Genus will have a Family foreign key that will point to Family’s entity primary key.

*Cardinalities: Every Genus will have only one family. Every Family can have at minimum one Genus but can be reference multiple genus entities.*

**Relations**

All of our entities and relationships, with the exception of the multivariable Fossil Author and Ref Author relations, moved directly to a relation in 3NF in a very uninteresting way, as we had already made them into 3NF when we designed them. Namely, every nonprime attribute in our relations is fully functionally dependent on their primary keys and there are no nonprimes which are transitively dependent on the primary key.

*Relation 1: Fossil*

**

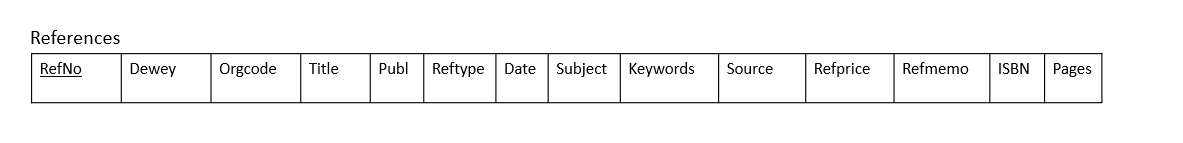
The Fossil relation is one of the most straightforward relations that we shifted from entity to relation. Fossil’s primary key is a super key for the entire relational model, however it only uniquely identifies the attributes listed in this relation. Each specific attribute listed here are unique for every fossil as they are mostly image data or miscellaneous information related to the fossil species.

*Relation : Fossil Author*



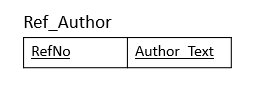
Fossil Author is a multi-varied attribute that relates the names of the people who discovered a fossil and named it to the fossil itself. Very often multiple people are credited with the discovery and authorship of a fossil genus and species, so we had to use ImageID and Name to uniquely identify the records in this relation. ImageID connects this relation to the Fossil relation. The primary key for this relation is a combination of the only two attributes, so obviously this uniquely identifies everything.

*Relation : References*



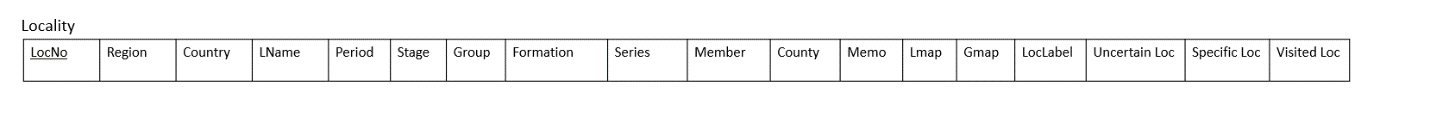
The References entity identifies the information about specific texts. The primary key RefNo uniquely identifies all other attributes inside of this relation, as RefNo refers to a specific book ID and each of these attributes are descriptive of this single book. References shares a 1 to N relationship to Fossil and Ref Author is a multi-varied attribute of References. This means that for the Fossil relationship, RefNo inside of References needs to refer to RefNo in Fossil. For the Ref Author relationship, the RefNo from Ref author refers to References’ Ref Author.

*Relation : Ref Author*



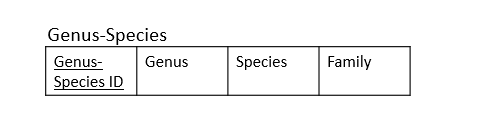
The Ref Author relation is representative of a multi-varied attribute from the References entity. The foreign key RefNo from the References entity and the attribute Name, that refers to the name of the author that wrote the reference, combined make up the primary key that uniquely defines this relation.

*Relation : Locality*



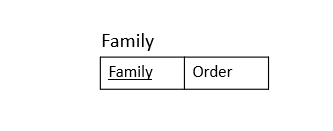
The Locality relation is uniquely defined by it’s primary key LocNo, which connect it to the Fossil relation. All the nonprime attributes within Locality are descriptive of the location that LocNo refers to, so the primary key LocNo defines all of them. When we created the Locality entity it was already in 3NF, so transforming it from an entity to a relation was trivial. The Locality relation shares a 1 to N relation with Fossil, which describes that each individual fossil was collected at a single location but a single location provided multiple fossils.

*Relation : Genus*



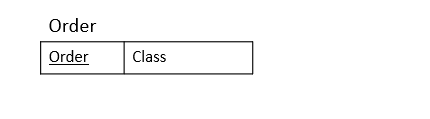
The Genus entity caused some trouble when we created it. Initially, we used Genus as a foreign key from Fossil, because I had thought that a Family in taxonomy uniquely identified a particular Genus. This turned out to be false, many Genus appear in many different Families. So we decided to use the binomial nomenclature of genus and species that is unique to all taxonomy in the form of a Genus-Species ID sent over from Fossil. In doing this, we found that we could refer to any fossil and get it’s specific Family which meant that the relations following Family would also return the higher taxa. Genus has a 1 to N relationship with Fossil and a 1 to N relationship with Family. This is indicative of the relationship of a genus and species within a Family, a single genus and species exists only within one family, which exists in one order, and so on. The GS-ID uniquely defines the nonprime attributes of this relation.

*Relation : Family*



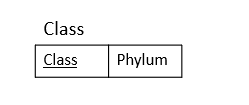
The Family relation has a 1 to 1 relationship with Genus and a 1 to N relationship with Order. This details the relationship of there being one Family for Genus and many Families in an Order. Converting this entity to a relation was trivial.

*Relation : Order*



The Order relation has a 1 to 1 relationship with Family and a 1 to N relationship with Class. This details the relationship of there being one Order for a Family and many Orders in a Class. Converting this entity to a relation was trivial.

*Relation : Class*



The Class relation has a 1 to 1 relationship with Order. This details the relationship of there being many Order in a Class. Converting this entity to a relation was trivial.

# Data Dictionary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Entity Name** | **Attribute** | **Data Type & Length** | **Primary Key** | **Foreign Key** | **Constraints** |
| **FOSSIL** | ImageID | Int(11) | Yes | Fossil\_Author(ImageID) | Incrementing number for each new record |
|  | Loc-No | Varchar(15) |  | Locality(LocNo) | Country code, hyphen, incrementing number |
|  | Taxacode | Varchar(50) |  |  | outdated |
|  | GS-ID | Varchar(7) |  | Genus(GS-ID) | Incrementing number for each new record |
|  | RC/Reference Collection | Int(11) |  |  | positive |
|  | XC/Extra Collection | Int(11) |  |  | positive |
|  | Special | VarChar(100) |  |  | TRUE/FALSE |
|  | PH\_Numbers | Varchar(35) |  |  | outdated |
|  | Common Name | VarChar(100) |  |  | commonly referred to as |
|  | TxtImageName | VarChar(100) |  |  | Image file path from C:/ |
|  | Photo | Varchar(100) |  |  | Image file path from C:/ larger image |
|  | Type | Varchar(200) |  |  | description of fossil type |
|  | RefNo | Int(11) |  |  | positive |
|  | RefPGFig | Varchar(20) |  |  | Image file path from C:/ smaller picture |
|  | Size | Varchar(20) |  |  | size of fossil in mm |
|  | Price | Varchar(20) |  |  | Dollar value |
|  | Figure | Varchar(100) |  |  | Image file path from C:/ of fossil in box |
|  | NumSpecimen | Varchar(20) |  |  | 10- or 10+ to show above or below 10 |
|  |  |  |  |  |  |
| **REFERENCES** | RefNo | Int(11) | Yes | References | positive |
|  | Dewey | Varchar(20) |  |  | dewey decimal: 3 digits, a decimal, 3 digits |
|  | Orgcode | Varchar(10) |  |  | Acronym for organization |
|  | Title | Varchar(200) |  |  | reference title string |
|  | Publ | Varchar(100) |  |  | Publisher of text |
|  | Pages | Varchar(35) |  |  | page numbers, sometimes a range of pages separated by a hyphen |
|  | Reftype | Varchar(35) |  |  | type of reference, ie website, journal, text |
|  | Date | Int(11) |  |  | year published |
|  | Subject | Varchar(35) |  |  | reference subject |
|  | Keywords | Varchar(100) |  |  | string |
|  | Source | Varchar(35) |  |  |  |
|  | Refprice | Double |  |  | reference price in dollars |
|  | Refmemo | Varchar(150) |  |  | string |
|  | ISBN | Varchar(35) |  |  | ISBN code, 1 digit, hyphen, 4 digits, hyphen, 4 digits, hyphen, 1 digit |
|  |  |  |  |  |  |
| **LOCALITY** | LocNo | Varchar(15) | Yes | Fossil(Loc-No) | Country code, hyphen, incrementing number |
|  | Region | Varchar(10) |  |  | country code |
|  | County | Varchar(35) |  |  | US county |
|  | Lname | Varchar(35) |  |  | string, common name for location |
|  | Period | Varchar(35) |  |  | string, age |
|  | Stage | Varchar(35) |  |  | string |
|  | Group | Varchar(35) |  |  | string |
|  | Formation | Varchar(35) |  |  | string |
|  | Series | Varchar(35) |  |  | string |
|  | Member | Varchar(35) |  |  | string |
|  | Country | Varchar(35) |  |  | string |
|  | Memo | Varchar(400) |  |  | string |
|  | Lmap | Varchar(35) |  |  | Image file path from C:/ large image of map |
|  | Gmap | Varchar(35) |  |  | Image file path from C:/ zoomed in image of map |
|  | LocLabel | Varchar(50) |  |  | string |
|  | Uncertain Loc | Varchar(35) |  |  | T/F |
|  | Specific Loc | Varchar(35) |  |  | T/F |
|  | Visted Loc | Varchar(35) |  |  | T/F |
|  |  |  |  |  |  |
| **Family** | Family | Varchar(100) | Yes | Genus(Family) | String, unique |
|  | Order | Varchar(100) |  |  | String |
|  |  |  |  |  |  |
| **Order** | Order | Varchar(100) | Yes | Family(Order) | String, unqiue |
|  | Class | Varchar(100) |  |  | String |
|  |  |  |  |  |  |
| **Genus** | GS-ID | Varchar(7) | Yes | Fossil(GS-ID) | String, unique |
|  | Genus | Varchar(50) |  |  | String |
|  | Species | Varchar(50) |  |  | String |
|  | Family | Varchar(100) |  |  | String |
|  |  |  |  |  |  |
| **Class** | Class | Varchar(100) | Yes | Order(Class) | String, unique |
|  | Phylum | Varchar(35) |  |  | String |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| **Fossil\_Author** | ImageID | Int(11) | Yes | Fossil(ImageID) | Incrementing number for each new record, unique |
|  | Author | Varchar(25) | Yes |  | String, unique |
|  |  |  |  |  |  |
| **Ref\_Author** | Author\_Text | Varchar(25) | Yes | References(RefNo) | String, unique |
|  | RefNo | Int(11) | Yes |  | Incrementing integer, unique |

# 

# Implementation

# 

Functional Requirement Construction

1. **The user must be able to get full taxonomic information for each unique fossil**
   1. To complete this requirement, we had to do a series of joins as the data required to meet the requirement was expansive. We had to traverse many relationships to achieve the goal.
   2. SELECT f.ImageID, g.GENUS, g.SPECIES, fa.FAMILY, o.ORDER, c.CLASS  
      FROM fossil f, genus g JOIN family fa JOIN orderfs o JOIN class c  
      WHERE f.GSID=g.GSID AND g.FAMILY = fa.FAMILY AND fa.ORDER = o.ORDER AND o.CLASS=c.CLASS  
      ORDER BY f.ImageID;
2. **The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils found in a specific location**
   1. To allow the user to view the proper data points, we had to use a join on fossil and locality. The country field on the query allows the user to enter the country they require to search from.
   2. SELECT PHOTO, GENUS, SPECIES, COUNTRY  
      FROM fossil f JOIN locality l, genus  
      WHERE f.LOCNO=l.LOCNO AND COUNTRY="BELGIUM"  
      ORDER BY PHOTO;
3. **The user needs to be able to view the larger images side by side with another larger image and some associated information (such as family, genus, species, location, formation, taxacode, locno) about both of them for identification comparison.**
   1. To compare multiple images, the user would need to run the query multiple times and compare the results
   2. SELECT f.PHOTO Photo1, fa.FAMILY, g.GENUS, g.SPECIES, l.COUNTRY, FORMATION, TAXACODE, f.LOCNO  
      FROM locality l, fossil f, genus g JOIN family fa JOIN orderfs o JOIN class c  
      WHERE f.GSID=g.GSID AND f.LOCNO=l.LOCNO AND g.FAMILY = fa.FAMILY   
      AND fa.ORDER = o.ORDER AND o.CLASS=c.CLASS  
      AND f.PHOTO LIKE "%S28.jpg"  
      ORDER BY f.PHOTO;
4. **The user needs to be able to print labels for any specific fossil or a group of fossils with information in the database. The labels have the following information: Genus, Species, Phylum, Class, Period, Formation, Location, Taxacode, and LocNo**
   1. The query required a significant number of joins to achieve the required result.
   2. SELECT GENUS, SPECIES, PHYLUM, c.CLASS, PERIOD, FORMATION, COUNTRY, TAXACODE, f.LOCNO  
      FROM locality l, fossil f, genus g JOIN family fa JOIN orderfs o JOIN class c  
      WHERE f.GSID=g.GSID AND f.LOCNO=l.LOCNO AND g.FAMILY = fa.FAMILY   
      AND fa.ORDER = o.ORDER AND o.CLASS=c.CLASS;
5. **The user needs to be able create or view “sets” of arbitrarily selected fossils for quick access to images and associate names**
   1. This query was designed to allow the user to design their search based on refererence number.
   2. SELECT ImageID,`COMMON NAME`,PHOTO,REFNO  
      FROM fossil  
      WHERE REFNO="25";
6. **The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils that were identified through a specific refno**
   1. To meet the user requirements, we based the searchability around the Reference Number of the dataset.
   2. SELECT PHOTO,GENUS,SPECIES,REFNO  
      FROM fossil JOIN genus  
      WHERE fossil.GSID=genus.GSID AND fossil.REFNO="153";
7. **The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils from the same formation**
   1. This query was designed to allow user to easily search based on formation type.
   2. SELECT PHOTO,GENUS,SPECIES,FORMATION  
      FROM fossil f JOIN genus g, locality l  
      WHERE f.GSID=g.GSID AND f.LOCNO=l.LOCNO AND l.FORMATION="RICHMOND";
8. **The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils in the same Phylum, Class, Order, Family, or Genus**
   1. This query was designed for the user to enter the phylum, class, order, family, or genus they desire, while commenting out the unnecessary search criteria.
   2. SELECT PHOTO,GENUS,SPECIES  
      FROM fossil f JOIN genus g JOIN family fa JOIN orderfs o JOIN class c   
      WHERE f.GSID=g.GSID AND g.FAMILY = fa.FAMILY AND fa.ORDER = o.ORDER   
      AND o.CLASS=c.CLASS  
      AND o.ORDER="MESOGASTROPOD"  
      #AND c.PHYLUM = ""  
      #AND c.CLASS = ""  
      #AND fa.FAMILY = ""  
      #AND g.GENUS = ""  
      ;
9. **The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils where the “Special” attribute is True to check which fossils are not currently in the physical collection but rather on display elsewhere.**
   1. This query is designed to allow the user to easily enter a value for special, without the need for modifying other portions of the query
   2. SELECT PHOTO,GENUS,SPECIES  
      FROM fossil f JOIN genus g  
      WHERE f.GSID=g.GSID  
      AND f.SPECIAL="true";
10. **The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils from the same period**
    1. To meet the user requirements, we configured the query to allow the user to easily modify just the period value without the need to modify any other portions of the query
    2. SELECT PHOTO,GENUS,SPECIES  
       FROM fossil f JOIN genus g, locality l  
       WHERE f.GSID=g.GSID AND f.LOCNO=l.LOCNO  
       AND l.PERIOD="MISSISSIPPIAN";

# Summary

The purpose of this project was to create a relational database that can be used to access identification information for fossils from the collection of Mike Bruggeman. The client had created an excel table that contained information to help in identification and reference for his own and his friends fossils, but he had always wanted to convert it into a database that he could eventually let people access all over the world. The this end, our group first gathered data and functional requirements from the client, divided the table into 3NF entities, then identified primary and foreign key constraints. We then created an ER diagram of the relationships between entities, established ordinalities, and detailed their relationships. After establishing this, we realized there were a few problems with our original entity creation, such as the fact that some Genui existed in several Families and that our author entities needed to be multivariate attributes- so we went back and recreated them. Finally, we were able to create our relational model and data dictionary- after which finished the MySQL implementation. We are now able to successfully query a MySQL database for the client’s many requested functional requirements.

Mike Bruggeman has one of the largest and most well cataloged private collections in the Southeast US. To his knowledge, there isn’t one that compares to his for the areas that he has collected. This is significant because the creation of an easily accessible database for the identification of the many fossils that a person can find all over the world will help many future generations once it is accessible. The work that our group has done was a good first step towards reaching that goal.

**Teamwork**

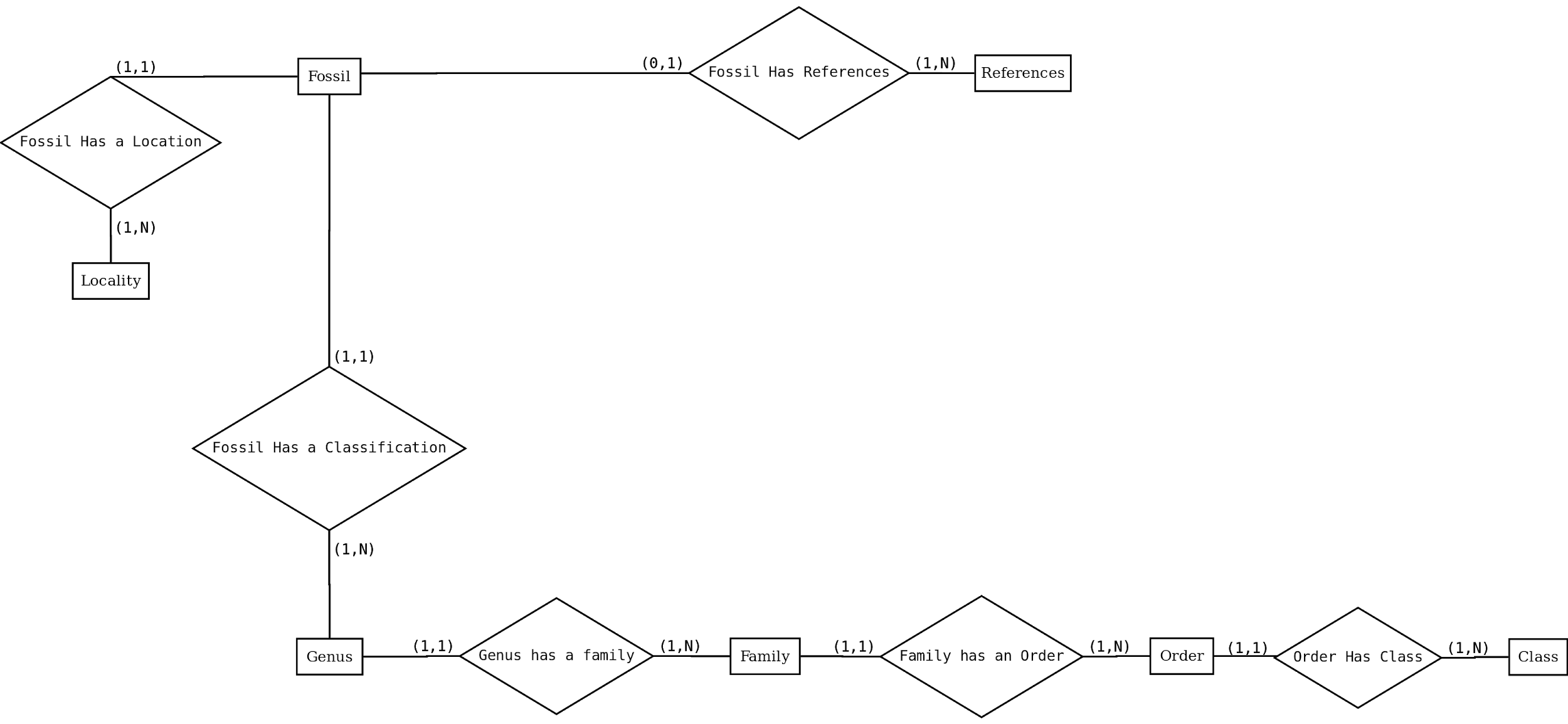
* *Jeff*
  + *Initial table creation*
  + *Data analysis*
  + *Requirements Analysis*
  + *Entity and attribute descriptions*
  + *Introduction*
  + *Relation descriptions*
  + *Data Cleansing/Formating*
  + *Summary*
  + *Commited changes to Final document*
  + *Presentation*
  + *Final document review*
* *Kevin*
  + *Relationship Diagrams and analysis*
  + *Document review and editing*
  + *Requirements Analysis*
  + *Created create.sql:*
  + *Created insert.sql:*
  + *Created queries.sql:*
  + *Implementation part of the Document*
  + *Presentation*
* *Julio*
  + *Relationship Diagrams*
  + *Entity Diagrams*
  + *Document editing*
  + *EER Diagram*
  + *Mapping from the EER model to the Relational Model*
  + *Relational model Diagrams*
  + *Final Relation model Diagram*
  + *Wrote the changelog and did upkeep to the changelog*
  + *Updated the Entities diagram and Relationship diagrams due to changes*
  + *Updated EER model after changes*
  + *Maintained and updated changelog*
  + *Final document review*
  + *Presentation*
* *Philipos*
  + *Data Dictionary*
  + *Presentation template choice*

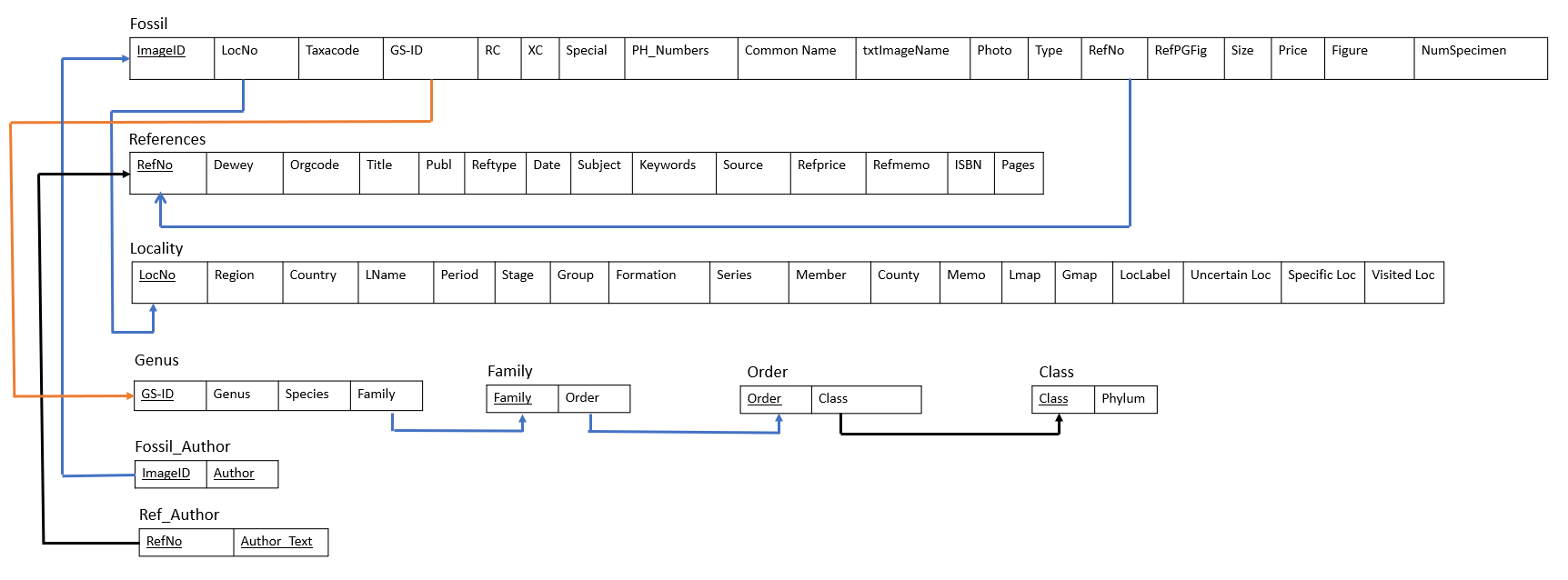
# 

# Appendix A

# 

EER Diagram



Relational model diagram

**Functional Requirements Queries:**

1. # Functional Requirement 1: The user must be able to get full taxonomic information for each unique fossil

SELECT f.ImageID, g.GENUS, g.SPECIES, fa.FAMILY, o.ORDER, c.CLASS

FROM fossil f, genus g JOIN family fa JOIN orderfs o JOIN class c

WHERE f.GSID=g.GSID AND g.FAMILY = fa.FAMILY AND fa.ORDER = o.ORDER AND

o.CLASS=c.CLASS

ORDER BY f.ImageID;

2) # Functional Requirement 2: The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils found in a specific location

SELECT PHOTO, GENUS, SPECIES, COUNTRY

FROM fossil f JOIN locality l, genus

WHERE f.LOCNO=l.LOCNO AND COUNTRY="BELGIUM"

ORDER BY PHOTO;

3) # Functional Requirement 3: The user needs to be able to view the larger image side by side with another larger image and some associated information (such as family, genus, species, location, formation, taxacode, locno) about both of them for identification comparison.

SELECT f.PHOTO Photo1, fa.FAMILY, g.GENUS, g.SPECIES, l.COUNTRY, FORMATION, TAXACODE, f.LOCNO

FROM locality l, fossil f, genus g JOIN family fa JOIN orderfs o JOIN class c

WHERE f.GSID=g.GSID AND f.LOCNO=l.LOCNO AND g.FAMILY = fa.FAMILY AND

fa.ORDER = o.ORDER AND o.CLASS=c.CLASS

AND f.PHOTO LIKE "%S28.jpg"

ORDER BY f.PHOTO;

4) # Functional Requirement 4: The user needs to be able to print labels for any specific fossil or a group of fossils with information in the database- the labels have the following information: Genus, Species, Phylum, Class, Period, Formation, Location, Taxacode, and LocNo

SELECT GENUS, SPECIES, PHYLUM, c.CLASS, PERIOD, FORMATION, COUNTRY, TAXACODE, f.LOCNO

FROM locality l, fossil f, genus g JOIN family fa JOIN orderfs o JOIN class c

WHERE f.GSID=g.GSID AND f.LOCNO=l.LOCNO AND g.FAMILY = fa.FAMILY AND

fa.ORDER = o.ORDER AND o.CLASS=c.CLASS;

6) # Functional Requirement 6: The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils that were identified through a specific refno

SELECT PHOTO,GENUS,SPECIES,REFNO

FROM fossil JOIN genus

WHERE fossil.GSID=genus.GSID AND fossil.REFNO="153";

7) # Functional Requirement 7:The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils from the same formation

SELECT PHOTO,GENUS,SPECIES,FORMATION

FROM fossil f JOIN genus g, locality l

WHERE f.GSID=g.GSID AND f.LOCNO=l.LOCNO AND l.FORMATION="RICHMOND";

8) # Functional Requirement 8:The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils in the same Phylum, Class, Order, Family, or Genus

SELECT PHOTO,GENUS,SPECIES

FROM fossil f JOIN genus g JOIN family fa JOIN orderfs o JOIN class c

WHERE f.GSID=g.GSID AND g.FAMILY = fa.FAMILY AND fa.ORDER = o.ORDER AND

o.CLASS=c.CLASS

AND o.ORDER="MESOGASTROPOD"

#AND c.PHYLUM = ""

#AND c.CLASS = ""

#AND fa.FAMILY = ""

#AND g.GENUS = ""

;

9) # Functional Requirement 9:The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils where the “Special” attribute is True to check which fossils are not currently in the physical collection but rather on display elsewhere.

SELECT PHOTO,GENUS,SPECIES

FROM fossil f JOIN genus g

WHERE f.GSID=g.GSID

AND f.SPECIAL="true";

10) # Functional Requirement 10: The user needs to be able to view the smaller images of fossils and associated genus and species based on a grouping of fossils from the same period

SELECT PHOTO,GENUS,SPECIES

FROM fossil f JOIN genus g, locality l

WHERE f.GSID=g.GSID AND f.LOCNO=l.LOCNO

AND l.PERIOD="MISSISSIPPIAN";

**Create SQL Code:**

DROP TABLE IF EXISTS `class`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `class` (

`CLASS` varchar(100) NOT NULL,

`PHYLUM` varchar(35) DEFAULT NULL,

PRIMARY KEY (`CLASS`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `family`

DROP TABLE IF EXISTS `family`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `family` (

`FAMILY` varchar(100) NOT NULL,

`ORDER` varchar(100) DEFAULT NULL,

PRIMARY KEY (`FAMILY`),

KEY `ORDER\_idx` (`ORDER`),

CONSTRAINT `ORDER` FOREIGN KEY (`ORDER`) REFERENCES `orderfs` (`ORDER`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `fossil`

DROP TABLE IF EXISTS `fossil`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `fossil` (

`ImageID` int(11) NOT NULL,

`GSID` varchar(7) DEFAULT NULL,

`REFNO` int(11) DEFAULT NULL,

`LOCNO` varchar(15) DEFAULT NULL,

`TAXACODE` varchar(50) DEFAULT NULL,

`PH\_NUMBERS` varchar(35) DEFAULT NULL,

`REFPGFIG` varchar(20) DEFAULT NULL,

`RC` int(11) DEFAULT NULL,

`XC` int(11) DEFAULT NULL,

`txtImageName` varchar(100) DEFAULT NULL,

`SIZE` varchar(20) DEFAULT NULL,

`TYPE` varchar(200) DEFAULT NULL,

`PRICE` varchar(20) DEFAULT NULL,

`COMMON NAME` varchar(50) DEFAULT NULL,

`SPECIAL` varchar(100) DEFAULT NULL,

`PHOTO` varchar(100) DEFAULT NULL,

`FIGURE` varchar(100) DEFAULT NULL,

`NumSpecimen` varchar(20) DEFAULT NULL,

`SET\_ID` varchar(20) DEFAULT NULL,

PRIMARY KEY (`ImageID`),

KEY `Loc-No\_idx` (`LOCNO`),

KEY `RefNo\_idx` (`REFNO`),

KEY `LOCNO` (`LOCNO`),

KEY `REFNO` (`REFNO`),

KEY `GS-ID\_idx` (`GSID`),

CONSTRAINT `GS-ID` FOREIGN KEY (`GSID`) REFERENCES `genus` (`GSID`),

CONSTRAINT `LOCNO` FOREIGN KEY (`LOCNO`) REFERENCES `locality` (`LOCNO`),

CONSTRAINT `REFNO` FOREIGN KEY (`REFNO`) REFERENCES `references` (`﻿REFNO`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `fossil\_author`

DROP TABLE IF EXISTS `fossil\_author`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `fossil\_author` (

`ImageID` int(11) NOT NULL,

`AUTHOR` varchar(25) NOT NULL,

PRIMARY KEY (`ImageID`,`AUTHOR`),

CONSTRAINT `ImageID` FOREIGN KEY (`ImageID`) REFERENCES `fossil` (`ImageID`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `genus`

DROP TABLE IF EXISTS `genus`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `genus` (

`GSID` varchar(7) NOT NULL,

`SPECIES` varchar(50) DEFAULT NULL,

`GENUS` varchar(50) DEFAULT NULL,

`FAMILY` varchar(100) DEFAULT NULL,

PRIMARY KEY (`GSID`),

KEY `FAMILY\_idx` (`FAMILY`),

CONSTRAINT `FAMILY` FOREIGN KEY (`FAMILY`) REFERENCES `family` (`FAMILY`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `locality`

DROP TABLE IF EXISTS `locality`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `locality` (

`LOCNO` varchar(15) NOT NULL,

`REGION` varchar(10) DEFAULT NULL,

`LNAME` varchar(35) DEFAULT NULL,

`PERIOD` varchar(35) DEFAULT NULL,

`LOCLABEL` varchar(50) DEFAULT NULL,

`COUNTRY` varchar(35) DEFAULT NULL,

`COUNTY` varchar(35) DEFAULT NULL,

`SERIES` varchar(35) DEFAULT NULL,

`STAGE` varchar(35) DEFAULT NULL,

`GROUP` varchar(35) DEFAULT NULL,

`FORMATION` varchar(35) DEFAULT NULL,

`MEMBER` varchar(35) DEFAULT NULL,

`UNCERTAIN LOC` varchar(35) DEFAULT NULL,

`SPECIFIC LOC` varchar(35) DEFAULT NULL,

`VISITED LOC` varchar(35) DEFAULT NULL,

`LMAP` varchar(35) DEFAULT NULL,

`GMAP` varchar(35) DEFAULT NULL,

`MEMO` varchar(400) DEFAULT NULL,

PRIMARY KEY (`LOCNO`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `orderfs`

DROP TABLE IF EXISTS `orderfs`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `orderfs` (

`ORDER` varchar(100) NOT NULL,

`CLASS` varchar(100) DEFAULT NULL,

PRIMARY KEY (`ORDER`),

KEY `CLASS\_idx` (`CLASS`),

CONSTRAINT `CLASS` FOREIGN KEY (`CLASS`) REFERENCES `class` (`CLASS`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `ref\_author`

DROP TABLE IF EXISTS `ref\_author`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `ref\_author` (

`REFNO` int(11) NOT NULL,

`AUTHOR\_TEXT` varchar(25) NOT NULL,

PRIMARY KEY (`REFNO`,`AUTHOR\_TEXT`),

CONSTRAINT `MyRef` FOREIGN KEY (`REFNO`) REFERENCES `references` (`﻿REFNO`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `references`

DROP TABLE IF EXISTS `references`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `references` (

`﻿REFNO` int(11) NOT NULL,

`DEWEY` varchar(20) DEFAULT NULL,

`ORGCODE` varchar(10) DEFAULT NULL,

`TITLE` varchar(200) DEFAULT NULL,

`SUBJECT` varchar(35) DEFAULT NULL,

`KEYWORDS` varchar(100) DEFAULT NULL,

`AUTHOR\_TEXT` varchar(100) DEFAULT NULL,

`DATE` int(11) DEFAULT NULL,

`PUBL` varchar(100) DEFAULT NULL,

`PAGES` varchar(35) DEFAULT NULL,

`SOURCE` varchar(35) DEFAULT NULL,

`REFPRICE` double DEFAULT NULL,

`REFMEMO` varchar(150) DEFAULT NULL,

`REFTYPE` varchar(35) DEFAULT NULL,

`ISBN` varchar(35) DEFAULT NULL,

PRIMARY KEY (`﻿REFNO`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

**Changelog:**

1. In the fossil entity, dropped author attribute.
2. In the fossil entity, added Fossil\_Author as a multi-valued attribute.
3. In the fossil entity:Relationships, changed from Family entity to Genus entity.
4. In the fossil entity, dropped Fossil has an Author relationship since it became a multi-valued attribute.
5. In the fossil entity, added GS-ID attribute
6. In the fossil entity:Relationships, Dropped set entity.
7. Changed fossil entity image to reflect changes.
8. In the entity references changed Author\_Text to a multi-valued attribute.
9. Dropped Fossil has an Author relationship.
10. Dropped References has an Author relationship.
11. Dropped Fossil\_Author as Entity.
12. Dropped Ref Author as Entity.
13. Dropped Phylum entity
14. Added Genus entity.
15. Added Genus entity diagram.
16. In the entity Genus added attribute Family, Genus, Species, and GS-ID
17. Added Genus:Relationships to Fossil and Family
18. In Family Entity, dropped attribute Genus and Species
19. In Family Entity, added order as an attribute that is a foreign key to order entity
20. In Family Entity:Relationship, dropped the fossil relation.
21. In Family Entity:Relationship, added genus relationship.
22. Updated Family entity diagram to reflect changes.
23. In Order Entity, Dropped family,Common name, and FN attributes.
24. In Order Entity, added class attribute as a foreign key to class entity.
25. Updated Order entity diagram to reflect changes.
26. In Class Entity, Dropped order and OL attributes.
27. In class Entity, added phylum attribute.
28. In class Entity:Relationships, removed phylum relationship.
29. Updated Class entity diagram to reflect changes.
30. Dropped set Entity
31. Updated Fossil has a classification: Fossil now relates to Genus entity rather than family.
32. Updated Fossil has a classification description: It now Fossil has GS-ID as the foreign key to Genus Entity.
33. Dropped Fossil has set relationship since Set entity was dropped
34. Adde Genus has a Family relationship
35. Updated the EER diagram.
36. Updated Data Requirements to reflect changes