CSC 436
Assignment 3

Database Application Design

E-R Model Design:

1. List the entity sets and their attributes with primary keys underlined in your design.

Entities	**Attributes**
Language	language_id, language_name
Function Implementation	implementation_id, example, syntax, result, notes, language_id (Foreign Key), function_id (Foreign Key)
Structure Implementation	implementation_id, example, syntax, result, notes, language_id (Foreign Key), structure_id (Foreign Key)
Operator Implementation	implementation_id, symbol, notes, example, language_id (Foreign Key), operator_id (Foreign Key)
Function	function_id, function_name, category, description
Data Structure	structure_id, structure_name, category, description
Operator	operator_id, operator_name, category, description

2. List the relationships between different entity sets in your design.

Entity	**Relationship**	**Entity**
Language	Implemented In	Function Implementation
Language	Implemented In	Structure Implementation
Language	Implemented In	Operator Implementation
Function Implementation	Has	Function
Structure Implementation	Has	Data Structure

Operator Implementation	Has	Operator

3. Construct an E-R diagram for your database application. Use the following steps to design your database using the E-R model:

    - Start by identifying the essential sets to be included in your database.
        - Completed.
    - Choose relevant attributes representing the values to be captured in the database for each entity set.
        - Completed.
    - Formulate relationships sets among entities, addressing potential redundancy in attributes.
        - Completed.
    - Incorporate any necessary constraints (ie, relationship cardinalities, total/partial participation, descriptive attributes, weak entity sets).
        - Original design did not include participation or descriptive attributes, but it has now been updated.
        - Completed.

4. Provide a detailed explanation of the rationale behind your design choice. Highlight how each element contributes to the overall functionality of your database application.

Our database application is designed to provide quick command-line lookups for programming syntax across multiple languages. When we first began brainstorming the E-R diagram, we ran into a few problems. Most of the examples that were covered in the textbook and lectures were examples that involved single value entries per attribute. Although this was effective for illustrating and learning the E-R Model, it was somewhat difficult for us to establish how we would be storing complex data, like code examples, complicated syntax, or other language-specific nuances, based purely off the provided examples.

So, we attempted to create a few basic entities, such as functions and programming languages. We thought that if we could create a simple base case for a lookup, then we would be able to establish exactly what attributes would need to be tracked/stored in the database. However, this led us straight into our first database design pitfall. We began by considering the requirements to do a direct translation between Python and C++ for a simple function like print(). The idea was to use the function name “print()” as our primary key for unique lookups and then return additional attributes, such as syntax, arguments, or other necessary information for key differences between languages. But then we remembered that the print function in Python is very different from C++. This led to many questions, such as do we include the type handling differences for each language, do we include necessary import statements between languages, how much information do we include from the documentation, how much information is necessary to include for quick lookups/translations? Ultimately, this made us realize that a single function entity that uses function names as unique identifiers/primary keys could not possibly scale well enough to be able to be used across multiple programming languages.

Our second pitfall came as a direct result of our lack of our understanding of the E-R Model. Before learning the E-R Model, we assumed that most databases were similar to excel spreadsheets or key-value pairs stored in tables. This led us to incorrectly assume that since we were using function names as primary keys and since each function could have very different operations per programming language, that we would now need a new entity for each programming language. However, this obviously led to scalability and redundancy issues. We then considered nesting attributes for each language, but quickly realized it would create a very complicated database design that would make queries and storage unnecessarily tedious.

To address the redundancy issues and to create more scalable entities for better storage and retrieval, we created the entities: Language, Function/Structure/Operator Implementation, and Function/Data Structure/Operator. The main idea was to stop relying on function/data structure/operator names for primary keys and focus more on how to include only the necessary information for retrieval without conflicts. This forced us to consider exactly which attributes were needed to be able to uniquely store and retrieve each function, data structure, or operator. So, by separating each function, data structure, and operator by each's unique implementation and programming language, we were able to create weak entities that allowed for unique storage in the separate function, data structure, and operator entities.

The primary keys "language_id" and "implementation_id" provide unique identifiers per programming language and implementation entity. The foreign keys "language_id" and "function/structure/operator_id" ensure that each of the Function, Data Structure, and Operator entities can include similar, non-conflicting entries. In fact, by implementing descriptive attributes to relationships, we can even now further separate the exact same entries, for example the same function, by the programming language implementation version or the date the function was added. Ultimately, the solution to our original design problems came by considering how we would identify/retrieve the data that we would be storing, not by how we would be storing it.