ANTIMICROBIAL PEPTIDE EDITABLE DATABASE

George Konstantinidis
University of Rhode Island, geokon13@msn.com

ANTIMICROBIAL PEPTIDE EDITABLE DATABASE

BY

GEORGE KONSTANTINIDIS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

2015

MASTER OF SCIENCE IN COMPUTER SCIENCE

OF

GEORGE KONSTANTINIDIS

APPROVED:

Thesis Committee:

Major Professor:	Joan Peckham
	Lenore M. Martin
	Lisa DiPippo
	Nasser H. Zawia
	DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2015

Abstract

The idea behind the AMPed database is to create an annotated collection of information about antimicrobial peptides derived from several existing larger and heterogeneous databases with the goal of uniformity and coherence. It will be an open source bioinformatics tool that researchers will be able to use in order to find and download genomic sequences relevant to their research, as well as obtaining links to the original data source. This thesis will discuss the original AMPed concept, the database's conceptual (ER – Entity-Relationship) design, the design and implementation of the database tables, and the Bioparser software tool for importing the data into the database. Before this work Dr. Martin of the University of Rhode Island and her students began this project by creating a preliminary design of the database. In addition a tool to extract relevant data from PDB (Protein Data Bank) XML files was developed. This work was informative, but none of the previous solutions were ultimately used in this project. Along with the design of the AMPed database, a bioparser software tool to parse and import bulk data was built by our team. In the future, prediction tools that are tailored to the needs of the peptide research community will need to be developed, but this is outside of the scope of this thesis. Furthermore, the AMPed group is developing a secure web interface alongside the completion of the thesis. This web interface work is supervised by Dr. Martin and advised by George Konstantinidis.

Acknowledgments

My deepest gratitude goes to my advisor, Dr. Joan Peckham, for the support and patience she has invested in this project and for her continuous support and guidance throughout these two years. My full appreciation goes to Dr. Lenore Martin for bringing me into her lab group and believing that I could accomplish her original vision. I am grateful to both professors for showing me the world of Bioinformatics, the possibilities that exist, and the need of Computer Science researchers to push the boundaries of the field even further.

I would also like to thank Greg Gardner for creating one of the calculation tools necessary for the overall project design. In addition I want to acknowledge the effort and addition of the AMPed undergraduate group, Jesus Mata and Airel Sosa, for their feedback and insight troubleshooting the database.

Finally I would like to thank my friends and family for helping me through tough times, supporting me, and never giving up on me throughout this journey of great personal and professional development.

Table of Contents

List of Figures

I. Introduction

The motivating problem of this project is that biologists around the world use databases, and other tools like Excel or Matlab every day to manage data for research and experiments. That was the fuel to start thinking about how to create a single anti-microbial peptide database that would filter and compile public data from several existing and heterogeneous protein databases. One difficulty that arises is the lack of uniformity across these databases reflecting the diverse international community of researchers who contribute data to these archives, so naturally this data source heterogeneity became an important obstacle to overcome.

There are currently three major protein databases existing around the world: DDBJ (DNA Database of Japan), EMBL (European Molecular Biology Laboratory) and NCBI (National Center for Biotechnology Information). Based on these databases, and the specific areas of interest to peptide researchers, there is a need to uniformly filter and consolidate the data into one focused database suitable for daily use by peptide research groups. David Ryder and Lenore Martin consulted with Daniel Ducharme to produce the initial conceptual designs for a prototype of AMPed in another project during the completion of his Master's Thesis [1], but the database tables were never completed or implemented. After identifying data of interest to expert peptide researchers, the next step was to design and build a database that could be used for many years, and would incorporate all future updates. In the first phase of this project the original Entity-Relationship (ER) Diagrams were completed, and the next step was to map the design to database tables, and finally to build the database and develop the bioparser to populate it with data.

The purpose of ER diagrams is to conceptualize the data model, and work with the peptide researchers (clients) to fully explore the relationships between data entities. The ER model is then mapped to the database schema using relational tables to structure the data. While building the data model, a basic decision is if the database will be normalized or not. A normalized database must be in at least 3rd normal form, in order to assure freedom from the most common anomalies. The purpose of normalization is to assure that the duplication of data and mishandling while executing insert, update and delete statements will not compromise the data. Decomposition of entities to tables with cross references using foreign keys is used in this case to normalize the data and therefore to avoid scenarios that could give rise to anomalies in the data. The consequence of normalizing databases is penalties in performance needed to re-constitute data entities when the database is queried, but as noted above, the benefit is error free query results due to freedom from anomalies.

Although the preliminary conceptual design of David Ryder [1] was never mapped to tables or implemented, this first prototype design was discussed with Dr. Martin, and based on her input this team developed a new design. After consideration of the data files that would be parsed and implemented into AMPed, we decided to map the conceptual design to a normalized database and thereby protect the integrity of the data. Cross referenced tables and unique keys were designed to support the import and re-import of the data. There are many database warehouses that use similar ideas for information storage either about proteins or other kind of data, but here a more advanced means for populating and updating the database was needed. A software tool, we named the Bioparser, to support this approach was designed to handle very specific files with a particular structure useful to peptide research but its design is extensible.

We designed and developed the Bioparser tool as part of this thesis, after requirements gathering with Dr. Martin. It handles local file parsing as well as downloading files from multiple locations, and then importing them into the database. Because every data source file has a different format, all the parsing methods were hard-coded for the source databases (DDBJ, EMBL, NCBI). In the near future will be necessary to work to fix possible defects of the tool or enhance it according to the constantly evolving requirements. Currently it handles bulk downloads from five specific databases: Uniprot, PDB, LAMP, BACTIBASE, and PhytAMP.

The basic goals of this thesis are to tackle the following three problems. First, is to come up with the best approach to download the data from the different data sources. Second, after the data has been downloaded, we have to filter and then import the data into our AMPed database. Third, we have to support database updates that will be done in the future, and make sure that data integrity is kept. That is, manual annotation of the data by the AMPed team and the database administrator must to be supported. Fourth, since we are dealing with large amounts of unrelated source data, we need to find the quickest and most efficient way to filter and import that data into AMPed. Finally the problem of normalizing the biological data, whilst keeping all the above considerations in mind will be the final task.

The secure web interface is also being built now by another graduate student of Dr. Martin's, Tripti Garg, to make the database available for public use and easy access. Very specific (compiled) queries have to be written in order to pull the needed data from the interface, and in addition, software tools using the data to calculate peptide properties need to be developed. One prototype tool has already been created by Greg Gardner, an undergraduate researcher in the Martin lab. He has developed a Chou-Fasman[i] structure prediction module to be integrated with the AMPed web interface.

Since the idea of this project is to be openly available, usable, and modifiable by the public for research and educational purposes, it was built with open source software. We built the database on a Centos 5 Linux server, in a MySQL RDBMS. The coding language that we used for the bioparser tool was the Go language by Google, because of its easy to use packages and its similarity to C and C++. The web interface will be developed using PHP, and HTML. Additional tools will probably be in Python; future plans will be discussed in Chapter VI.

II. Related Work

There have been previous databases that handle antimicrobial peptides, but most of them are lacking in quantity, uniformity or freshness of data. The predominate archive that holds the most protein sequences and associated 3D structures is the PDB (Protein Data Bank) [2]. As of today PDB, developed and managed by RCSB (Research Collaboratory of Stuctural Bioinformatics), holds about 250,000 peptide sequences and about 97,000 protein structure files. It is used as the primary resource for all antimicrobial peptide database warehouses. The three collaborators of the RCSB are the University of Rutgers, the University of California San Diego, and the University of Wisconsin Madison. It is a very large repository that has more protein than peptide data, so for researchers studying antimicrobial peptides, this data needs to be extracted. It was decided that it is more efficient to first download all the data in the source DB, and then to filter and extract the relevant data subset with the Bioparser.

Another important database in the field of peptide research is UniProt/SwissProt (Universal Protein Resource sub DB of EMBL). The UniProt database currently holds about 540,000 curated protein entries. Again, like PDB, this database holds all known proteins, excluding synthetic sequences, most patent application sequences, small protein fragments predicted to be encoded by nucleotide sequences, pseudogenes, and most non-germline immunoglobulins and T-cell receptors.

One of the best known antimicrobial peptide databases, in the world of peptide biochemistry, is the APD (Antimicrobial Peptide Database) [3], and its newer version APD2 [3] from the University of Nebraska, maintained under the supervision of Guangshun Wang. This database is manually curated, and has selected entries derived from PDB, PubMed, Google and SwissProt. It only holds peptides meeting specific criteria, which are: if they are from natural sources, if their MIC is less than 100uM and if the peptide sequences contain less than 100 amino acid residues. Based on these criteria, the number of peptides that APD2 holds is currently 2,370. The update process for APD2 always takes a long time because it is manual and expert peptide scientists have to review all the entries one by one.

Finally, PhytAMP [6], LAMP [7] and BACTIBASE are also being imported to AMPed by the current implementation of the Bioparser. Some other databases that could be sourced in the future, by modifications to the existing Bioparser tool are DAMP, YADAMP and CAMP. All of these are databases that can be used to create a collection of antimicrobial peptides by using specific criteria to select and store their data post-download; most existing repositories dedicated to antimicrobial peptides import data from only two or three large databases, those typically being Uniprot or PDB. In addition, the most common problem with these specialized databases is the structure of the data and how it is being exported. For example PhytAMP and

BACTIBASE don’t store all the same data, nor do they use the same naming conventions. BACTIBASE holds the Cysteine, Glycine, Boman Index, Instability Index, etc data, while PhytAMP holds Catalytic activity, Polymorphism, Mass Spectrometry and other data. AMPed will try to selectively capture and sort all the data derived from these different source databases, over the progression of the project, and store them in one repository devoted to antimicrobial peptide research in a quick, efficient, and uniform way.

III. Initial Design, Tables, and Ideas

The goals of our project are to have a complete database with an import tool and to create various ways to import data while minimizing the involvement of the administrator. Looking at previous work, for example APD2, PhytAMP, DAMP, of other peptide databases, a common theme is the manual annotation of the work, and the not so frequent updates from the larger databases (PDB, UniProt). In order to provide a solution for these issues, different approaches were taken. One of them is to create a sanitizing database prior to the construction of the database in order to minimize error in the import process. The sanitizing database acts as a filter in order to catch any bad data (wrong format, wrong type etc.), and remove it or fix it before the final import. Also, prior to the sanitizing database we aim to extract all the data from the original files in CSV (comma separated values) format to make it easier to import. Another approach is to import everything directly from the files of the database of origin, and deal with the parsing while importing.

The first approach, though, had a couple of issues. First is the involvement of the database administrator to initiate the scripts one by one in order for all the data formatting, and import processes to be complete. Also, while researching the size of the files, and the quantity of the files that would have to be imported this process did not make much sense because of the time it would take for the whole cycle to complete. Thus this was addressed by switching to a scripting language that can implement more efficient parsing times and faster imports to the database. For error handling, a log file is kept during the execution of the data import cycle to record the location of errors. Afterwards the administrator can go back and trace the erroneous files one by one and import the data manually, once the database is up and running. For the smaller sized files the parsing and import process takes only a few minutes, while on the other hand for the PDB files which are approximately 96,000 structure files, the download time is about 15 hours, and the import process takes about 3 days with the current server and hardware. If in this process additional steps were added and more control given to the administrator the complete cycle would take over 7 days.

The UniProt file is 2.8 GB unzipped and 470MB zipped, so an approach for splitting the big data files was considered for reduced processing times and eliminating a bottleneck to the server. The choice though of the scripting language makes it possible for a zipped file to be parsed directly without unzipping it, and only loading in memory the part that is being parsed. What this means is that for every entry that the Bioparser tool would try to parse through, it would only load that string of text in memory and then immediately dump it for the next entry. This method does not overload memory of the server and assures fast processing times.

In addition to the import and parsing methods, the tables were database tables were redesigned. To support the way the Bioparser tool would work, and in order to minimize processing time, some tables were dropped and others were created. All these changes are mentioned just below.

An initial design of the E-R diagram, before our changes, is the following:

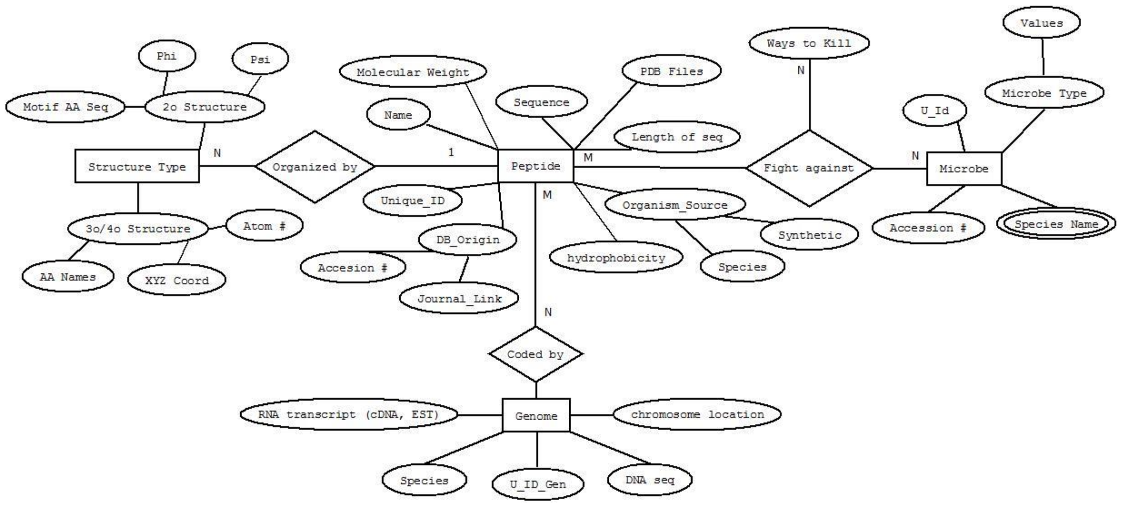


Figure 1. Initial design of E-R diagram of the AMPed database.

The complete design of the AMPed database was created entirely by the team and none of the designs from David Ryder was used. This was a lengthy process and one of the hardest parts of this thesis. In order for the computer scientist to understand the biology and the data model, many meetings were needed. The design kept changing as more information kept being added, but with constant communication and knowledge transfer the final design of the database was built. Some of the changes that were made along the way are described below.

One of the decisions that were made in order to accomplish some of our goals and to fulfill the needs of the group, was the addition of the Inserted_By flag across the different tables. With this flag in place we can control what the user will see as a manually annotated entry vs. an automatic import straight from the data source. In addition it helps with the update of the database, because when we run the Bioparser tool in order for it to import all the new data, it will first go in the AMPed database and delete all the existing data with an Inserted_By flag set to 1. That is done because the files that are given to us by the sources have in them all the pre-existing data plus all the new updates that they have made over the past few months. So in order to not have duplicate data within AMPed, we wipe the pre-existing data that have never been annotated by our team and re-import them.

There are several parts of the design that did not change; for example the Peptide table, which is the main table in the database, did not change. A few tables that carry important information are Microbe, Peptide_Structure (2nd Structure), Genome and 3rd_Structure. Minor changes were made on these. Additional changes were made to some of the cross reference tables, such as in Organized_By, in order to gain greater efficiencies and ease in querying the data. This is explained in more detail in the Tables section of this thesis.

Major changes include the addition of the following tables: Deleted, Inserted_By, and Journal. The third table was originally part of the Peptide table but this decomposition was needed to create a normalized database. Further detail will be given in the Tables section but the following sub diagram shows the connection between each of these tables and the main table and the Peptide table through the foreign key, Unique_ID.

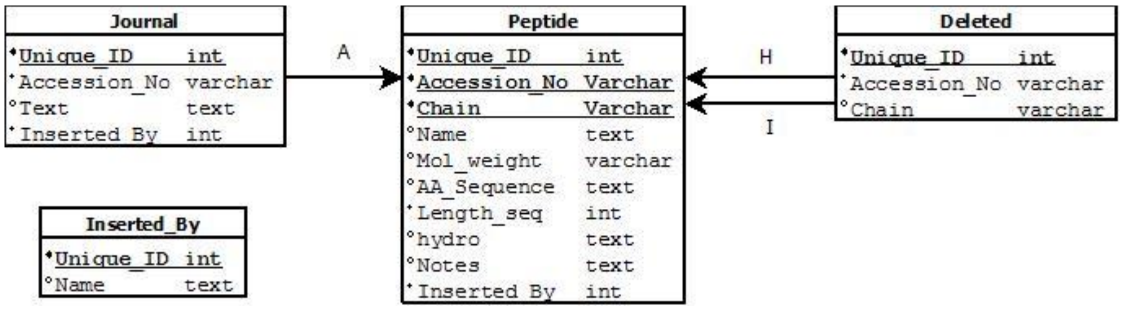


Figure 2. Sub diagram of the Deleted table, the Journal table and the Inserted By.

As another example, part of the complete diagram of tables is shown between the 3rd_Structure table and the Peptide table.

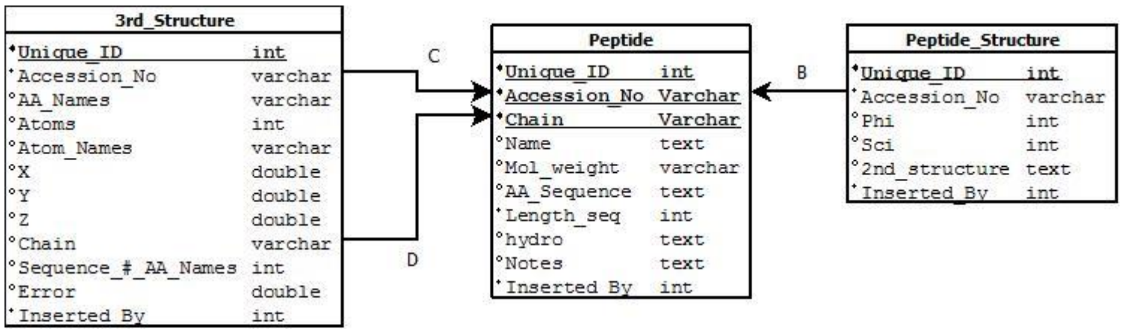


Figure 3. Sub diagram of the table structure between the 3rd_Structure table and the Peptide.

IV. The Final Layout

In this section the final tabular structure of AMPed is discussed. More over the concept of the whole design, the databases used, and the data that has been imported so far will be analyzed.

a. The databases

There are many databases [2-9] holding information that the scientists could utilize, but currently AMPed combines five of them. There are three small databases from which peptide data was initially extracted, the PhytAMP [6], LAMP [7], and BACTIBASE [9]. In addition two larger databases, UniProt[8] and PDB (Protein Data Bank) [2] were used. Each data file has a different structure, different data and somewhat different format. Most of them use a FASTA format, which is a text-based format for representing either nucleotide sequences or peptide sequences, in which nucleotides or amino acids are represented using single-letter codes. The format also allows for sequence names and comments to precede the sequences [10]. The differences among these data files is that some have an accession number, a peptide name and a sequence, while others hold more information. (An accession number in bioinformatics is a unique identifier given to a DNA or protein sequence record to allow for tracking of the different versions of that sequence record [11].)

This is why for each of those files have different methods of parsing the data in preparation for download to our database have been coded.

The first database that was chosen for study of its contents and structure was BACTIBASE [9]. The format of the data to extract was in FASTA, which made for easier parsing. There are 4 values that exist in the data file for each entry. The first value is the accession number of the database, and its unique identifier. The second value is the name of the anti-microbial peptide as it was assigned by the scientists. The third value contains notes that might accompany the specific entries. And finally the fourth value is the amino acid sequence that also unique for each peptide. BACTIBASE contains calculated or predicted physicochemical properties of 220 bacteriocins produced by both Gram-positive (196) and Gram-negative bacteria (19) [2]. This database also holds structural data in XML format that will either have to be uploaded manually or automatically extracted by a tool. This tool could be developed in the future to extract the information from the XML.

The second database selected for data extraction was LAMP [7]. LAMP’s data is also in FASTA format but this database holds more values than the previous one. The first value in most FASTA formats is the accession number of the database, and the second value is the peptide name. The third, notes, gives the source of the peptide, and then the fourth value explains the nature of the peptide as either from a natural source or synthetically made. The fifth value describes the source of the experiment as either natural, predicted, or synthetic, along with the peptide. The sixth value describes the type of microbe that the peptide fights against, for example anti-microbial or anti-fungal. The last value in each entry again is the amino acid sequence, which is always unique to each peptide.

Another similarly formatted database is PhytAMP [6]. Its entries are also in FASTA format and it holds only very basic information. The first value is the accession number, the second is the name of the peptide and the third is the amino acid sequence. PhytAMP is a database dedicated to plant antimicrobial peptides. The importance of plants is that they produce small cysteine-rich antimicrobial peptides as an innate defense against pathogens such as a-defensins, thionins, lipid transfer proteins (LTPs), cyclotides, snakins as well as hevein-like proteins [6]. Just like BACTIBASE, this database also has structural data in XML format in addition to the FASTA formatted file. The user will have to manually download the XML files and import them one by one, or design a tool in the future to extract them.

One of the bigger and well known databases that scientists use is UniProt [8]. The data file of this database is about 2.8 GB in size, uncompressed and 460 MB compressed. The UniProt files will be automatically updated every time the team decides there are new values for import. The data file that holds all the entries has a lot more information for each entry than any of the previously discussed data files, but only some of this information will be parsed and stored. The data fields that we downloaded from this database were: the accession number, the name of the peptide, the organism source, the organism host (species of the microbe), the journal or publication reference, and the amino acid sequence including the amino acid sequence length. Each entry in the file ends with a double forward slash "//". This makes it easier for us to know where the entry is complete, and move on to the next row for insertion. The double slash was not needed in the imported data, but only as a placeholder to signify the end of the entry. Due to normalization of the UniProt database, the information for each peptide will reside in multiple tables, so it will be extracted using relational queries and recomposed for insertion into the AMPed database.

The last database used in this project for importing anti-microbial peptide data is the PDB (Protein Data Bank) [2]. This data is split in two different files. The first is the pdb_seqres file that stores all the basic information in a FASTA format file but with different delimitation than other similar data files. The previous files were pipe delimited, while this data file is space delimited with abbreviated identifiers. The first value is the accession number but that is split into two different fields in the database because the second part (after an underscore) is the peptide chain. The second value, which is not parsed, states molecule type. We do not parse this data because all of the data is of type protein. The third value is the amino acid sequence length, which the database stores but does not import from this field; rather it calculates the sequence length in real time and stores the length in the appropriate field. This calculation method is used for greater accuracy in the case that the data that holds the sequence length is incorrect. The fourth value is the peptide name, and finally the last value is the amino acid sequence.

There is also a structure file that holds additional information about each entry in the first file. That file is also loaded into the database. All of the structure files can be found in a different FTP location with “.ent” file extension,[12]. As was mentioned earlier, each entry from the FASTA format file has the accession number and the chain as part of the name. This happens because each peptide holds three dimensional representations of the peptide and there can be multiple representations for the same peptide, thus this could cause confusion and duplicate unique entries in the database. We have used the accession number and the chain to make each entry, as well as the structure files, unique. The PDB structure files contain a large amount of information but again not all of this will be parsed and stored. The values that we have extracted are the accession number, the chain, the name of the peptide, the type of the peptide either natural or designed in a lab, the source of the peptide either designed or natural or synthetic, the journal reference, the secondary structure of the chain with the specific residues that are present, and finally the third structure of the peptide. The third structure holds a large volume of data and also has a 1-N relationship to the Peptide table. For each Accession_No there are thousands of third structure entries, so each has its own tuple and entry for each atom number. Each entry has the atom number, the atom name, the amino acid residue names, the chain, the sequence number of the amino acid names, the x, y, and z coordinates, the error (otherwise called the temperature factor), and the charge of the atom. These values together contain the information needed to build the third structure of the peptide. In order to query these entries from this table and link them to the rest of the data in the database, the query will use the accession number and the chain as the primary key identifier.

b. The Concept

As mentioned previously the whole concept of this project is to bring data from different databases together in a uniform and comprehensive way to support the needs of peptide researchers. In order to do so the team developed a design to capture all of the data that scientists need from different databases. A few of the tables have been analyzed above to show the design of their contents, but next we show the high level design.

The main table of concern as well as the central point of the database is the Peptide table. The initial discussion for this project started from that table and continued to include the other needed tables. When scientists search for a peptide they always look for its accession number, or for its amino acid sequence. Once they have found the peptide, then more information is usually displayed. Depending upon the database the scientists are looking at, they will see different data. For example the PDB database holds mostly the structure of the proteins. This is in comparison to UniProt which holds the types of microbes the peptide will be fighting against.

One of the main features of the AMPed database is that all the data is imported automatically using the Bioparser tool using the Inserted_By field. This is a field in each of the tables holding imported data. There is also a separate table named Inserted_By that is used as a reference for the user for the meaning of 0 and 1 (manual manipulation vs automatic). This field holds a value of 1 or 0 and is used to determine the difference between manually curated and bulk imported data. For each tuple that is imported with the Bioparser tool, the Inserted_By field is set to 1. For any entry that manually changed for any reason, the Inserted_By field is given a value of 0. In the next import process everything with a value of 1 gets deleted automatically, and all databases get re-imported. That is, all previously imported and un-curated data is refreshed, and all curated data entries are retained.

The Relationship between Peptides and Microbes

The Microbe table holds information about microbes. This table also holds information about the species of the host of the microbe. Because of the relationship between peptides and microbes, although they are represented in separate tables, we have put a cross reference table between them to provide referential integrity. This decomposition and cross referencing is the general principle of data normalization to prevent anomalies in queries. The table that connects the Peptide and Microbe tables, is Fight Against and it contains a primary key as well as information about the ways in which the microbes are fought by the peptides. The relationship between the Peptide and the Microbe table is M to N, meaning one peptide can fight against many microbes, and one microbe can be fought by many peptides. The primary key of the Peptide table is the Accession Number, and the primary key of the Microbe table is the Microbe ID. Below is a sub diagram of that relationship.

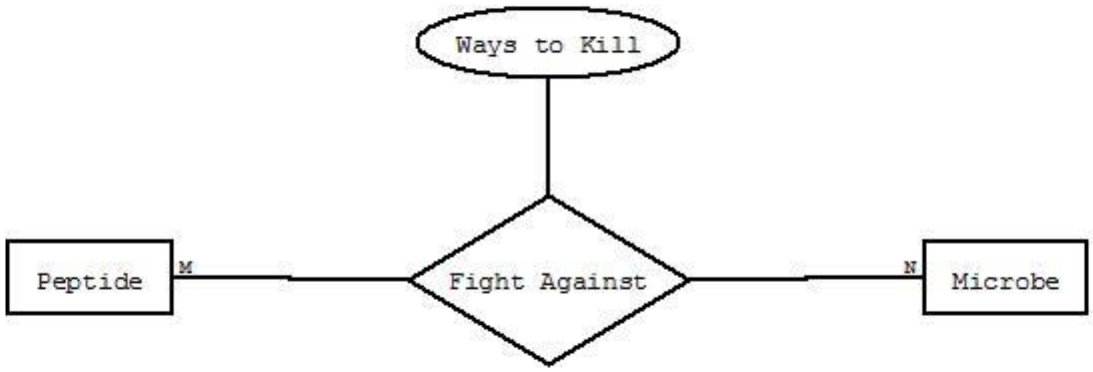


Figure 4. Sub diagram of the complete E-R diagram. Relationship of Peptide and Microbe table.

The Relationship between Peptides and their Genome

These small proteins, or peptides, are derived from either an organism or a peptide that is designed in a laboratory. The Genome table stores information about the host, the DNA sequence, and the chromosome location of each Peptide. The relationship between the two tables is M to N, meaning that each peptide could be found in many genomes, and each genome might have many peptides. For that reason, a cross reference table called Coded By holds the Accession Number from the Peptide table and the Genome ID from the Genome table. Below is a diagram showing that relationship.

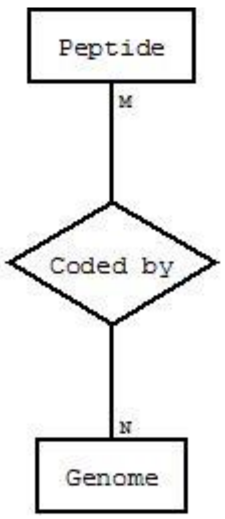


Figure 5. Sub diagram of the complete E-R diagram. Relationship of Peptide and Genome table.

The 2nd and 3rd Structure of a Peptide

All proteins can be described by different forms and structures. The amino acid sequence is the first structure and that is why it is stored in the main table, but there are also 2nd and 3rd structure descriptions. There is a table describing each of these structures. In the AMPed database the 2nd structure of a peptide is described through the Peptide Structure table. The secondary structure of a protein is a sub-group of residues, or sub-structure, of the original first structure that is classified as an alpha-helix, beta-sheet or turn [14]. Within that table we also store the Phi and Psi, angles that range from -180 to 180 degrees. These are also called the Ramachandran angles due to the Ramachandran plot that determines the secondary structure of a peptide. These will be calculated with separate tools on the front end web interface. The third structure of the peptide is the three-dimensional structure of a single, double, or triple bonded protein molecule. The alpha-helixes and beta pleated-sheets are folded into a compact globular structure. The folding is driven by the non-specific hydrophobic interactions, the burial of hydrophobic residues from water, but the structure is stable only when the parts of a protein domain are locked into place by specific tertiary interactions, such as salt bridges, hydrogen bonds, and the tight packing of side chains and disulfide bonds [19]. Those two tables each have 1 – N relationships with the Peptide, thus the primary key of the Peptide table is also used as a foreign key in each of the 2nd and 3rd structure tables. In the 3rd Structure table because the same accession number is used for many entries, the Chain field is included with the Peptide ID to form the primary key to assure unique values. Below is a diagram showing that relationship.

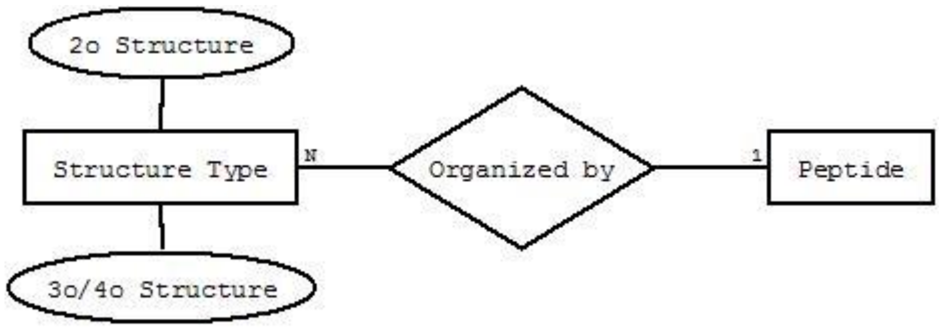


Figure 6. Sub diagram of the complete E-R diagram. Relationship of Peptide to 3rd Structure and Peptide Structure table.

The last table, the Journal table, holds data gathered from the data files. This table holds the reference to the research paper that created the entry in the databases from which the entry was imported. Most of these journal “links” also hold the PMID, which is the PubMed unique identifier for the publication. This is of great importance to scientists because they can refer back to the original publication to see how the results were gathered and which methods were used.

There are two tables in the database that describe the way that the data is stored and modified. The first table is the Deleted table. This acts as a place holder for all entries that need to be “deleted”. Eventually no data from the database will ever be deleted because in the next update of AMPed, all data will be re-imported. This table was created to simply hide the data from the front-end user. When the user decides to search for something specific, any entries that exist in the Deleted table will not appear in the search results of the user, based on specific queries that will be built into the interface. That will make the data “deleted” inaccessible. The second table that comes into play alongside the Deleted table is the Notes table. This table allows our team to modify data, by entering any needed comments or notes into its Text field. If, for example, we decide to “delete” the data by placing the entry in the Deleted table, we could also enter that entry in the Notes table and comment in the Text field why it was deleted. At the same time the Notes table, is linked to all the Inserted_By columns that exist in every table and automatically take the value 1. The database is currently set up to pull all data from the sources specified and upload everything that holds an amino acid sequence length of less than 100. Every time this data gets uploaded the Inserted_By value takes a value of 1, simply stating that it was uploaded “Automatic”. The same thing will also happen for every re-import. When the team decides that it is time to refresh the database it will wipe out all data from all tables that hold the value 1, and then reimport everything and set the Inserted_By value to 1 again. If we want to modify though an entry by entering some comments in the Notes table, the value of the Inserted_By field would be changed to 0 to assure that the Bioparser tool will not delete the annotations. It will of course reimport the same value but the front end user will be able to distinguish between the original entry which came from the data files, and the modified one by the AMPed users. This method of deleting everything and reimporting everything again has been adopted to make sure that the latest version of each entry is stored.

c. The tables

As mentioned above, we have made changes to the original AMPed tables and these changes include the following. The Organized_by table has been dropped because the relationship between the Peptide table and the 2nd_Structure is 1 – N. A foreign key, Accession #, from the Peptide table has been placed in 2nd_Structure in order to JOIN the two tables.

An expansion to the 3rd_Structure table has been made. Initially this table was to hold only 5 values, but at this moment it holds 11 values to include additional information needed for the researchers. This decision was made after extensive consultations with Dr. Martin on the content of the data files and the information that should be kept. The initial 5 values came from the prior designs of David Ryder [1]. This table has a unique key and foreign keys to other tables using Accession_# and Chain to provide access to other from the Peptide and Peptide_Structure table through joins.

The Peptide table

The main table, Peptide, controls the rest of the database and the links to the other tables. This table holds the basic information about the small proteins under 100 Amino Acid residues long called peptides. It has three candidate keys: the unique id, the accession number and the chain. The first one is the auto incremented primary key of the table, the second one is the Accession_# of the different databases, and the third one is the Chain of similar peptides that have unique entries due to the information extracted from the individual structure files. The 2nd and 3rd columns are the candidate keys and can be used in other tables as foreign keys. The unique id is the primary key. It signifies a distinct entry in the table and it is only used in the Fight against and in the Coded by table. Next is the Name of the Peptide, followed by the Molecular Weight which currently is empty because this value is not pulled from any files, but rather will be calculated using tools that will be developed. This will be calculated as part of a query when the web interface of the database is built. It will likely be incorporated within the PHP code. This column only exists in case the team decides in the future they would like to store the results that will be produced from the calculation, otherwise in future work it could be removed from the table. One of the most important columns in the database is the AA (amino acid) sequence that is stored in the Peptide table. This sequence it is used to determine if something is really a Peptide. Following this column is the Length of the AA sequence, which could be pulled from the data files, but instead it is to be calculated during parsing to avoid errors. The next two columns, also currently empty, are the Hydrophobicity and the Notes field. The first of those two fields will be calculated just like the Molecular Weight, it exists to store the calculated value. The latter is a field that would take in any value that might be of importance from any of the data files, but does not belong in another field. It is not currently used anywhere but with the expansion of the database it might be incorporated in later designs. The last field of this table, and also in every table, is a value of great importance in the process of importing data and updating the database, and that is the Inserted_By field. As described earlier, this field acts as a flag in every table. There will be special queries that will set this flag to 0 every time there is an update statement run, or a new entry is inserted manually. Eventually the database might hold at the same time two entries with similar information, but the difference will be the Inserted_By field. One will show a value of 1 and the other a value of 0. The scientists will know which ones remained the same since the import process, and which ones might hold additional information that Dr. Martin and her lab have discovered.

Table structure:

Attribute:	**Data type:**
Unique_ID	Int (10)
Accession_No	Varchar (15)
Chain	Varchar (15)
Name	Text
Mol_weight	Varchar (15)
AA_sequence	Text
Length_seq	Int (10)
hydro	Text
Notes	Text
Inserted_By	Int (11)

The Genome table

The next table in the AMPed database is the Genome table. The primary key is Genome_ID. The Genome table mostly holds information about the source of the Peptide and if it is from a specific kind of species or if it was created in a laboratory. One of the columns in the table is the original DNA (Amino Acid) sequence. Not all data files hold this information but it is required here for research purposes. There will be instances where null values will exist in the database, but those require minimal space of 2 bytes; the total size of the database currently doesn't exceed 3.5 GB. The next column is the Chromosome Location which is a single piece of coiled DNA containing many genes, regulatory elements and other nucleotide sequences [15]. Next is the Species of the Genome, which is basically the organism from which the peptide was extracted. Many times the peptides are not really taken from organisms but rather manufactured in a lab, and for that reason we have included a column named Source. In some cases where the peptide is not manufactured from a known species or an existing peptide, then biologists might create a completely new sequence. For this situation, we have created the Source_Extra column, specifying this distinction. The description of the RNA Transcript that is used as the next field in the Genome table is described as follows: The DNA contains the master plan for the creation of the proteins and other molecules and systems of the cell. Carrying out of this plan involves transfer of the relevant information to RNA in a process called transcription. "The RNA to which the information is transcribed is messenger RNA (mRNA)." [16]. And finally as mentioned above, the Inserted_By field is used across the database to distinguish the "automatic" imported data, from the "human" edited or imported data.

Because the relationship between the Peptide and Genome tables is M – N, a cross reference table is required for normalization purposes. The Coded_by table connects the two tables through their two primary keys, Accession Number from Peptide, and Genome_ID from Genome.

Table structure:

Attribute:	**Data type:**
Genome_ID	Int (11)
DNA_seq	text
Chromosome_loc	Varchar (255)
Species	Text
RNA_transcript	Text
Source	Text
Source_Type	Text
Inserted_By	Int (11)

The Microbe table

Continuing on with the role and the significance of the tables in the AMPed database, the next one is the Microbe table. By its name someone could easily understand that it is the microorganisms that the peptides are fighting against. It holds the basic information of all the microbes that the peptides are targeting, and will not always be populated because of the way the data files are handling this information. Out of the five databases that are being imported currently into AMPed, Uniprot does carry that information and thus we extract it and populate it, but for example LAMP and

BACTIBASE don't include that information in their data files and has to be left blank. The first field in this table is the ATCC_Accession_No, which stands for the American Type Culture Collection Accession Number. It is the unique identifier that is used globally by scientists to order samples of a specific protein or a genome for research purposes. The next field over is the Species Name that specifies the host/species of the microbe. Next, is the Microbe Type which designates what kind of microorganism it is, Viral, Fungi, Bacteria, Protist or Archaea. And lastly the Inserted_By field which has been described previously.

Although the Microbe table and all its contents are mentioned, the connection to the Peptide table was not mentioned, and that is why the Fight_Against table exists. The relationship between these two tables is an M – N, and for that reason we need for separate table. The main fields of this table are the foreign keys from the Peptide and the Microbe table. But in addition to that it also holds information about the specific method by which the microbes are killed by the peptides. Every method has its own field and thus Boolean values are used to determine if it is or isn't used. The three different methods are, In Vivo, Zone Inhibition, and Broth. The first method is used for testing the antimicrobial peptides directly on the living organism that hosts the microbe. The second method is used with an agar plate to control the growth of the microbe outside of the specified area on the plate. And the third method is used with a liquid nutrient mixture in order to grow the microbes and test them against the peptides [17,18].

Table structure:

Attribute:	**Data type:**
Microbe_ID	Int (10)
ATCC_Accession_No	Varchar (255)
Species_Name	text
Microbe_Type	Varchar (50)
Inserted_By	Int (11)

The Peptide Structure table

The following table holds information about the 2nd structure of the peptide. This table is the Peptide Structure table, and by saying 2nd structure it means identifying which part of the sequence is an alpha-helix, a beta-sheet or a turn. The first field in this is the Accession Number that comes from the Peptide table as a foreign key. The next two values are Phi and Psi angles and will be calculated from the Amino Acid sequence using calculation tools that will be developed. Currently all these values are empty but once they are calculated the team could decide to store them for each entry. The field that designates where in the sequence it becomes a helix, a sheet or a turn is the 2nd Structure field, and it is stored as text.

Table structure:

Attribute:	**Data type:**
Unique_ID	Int (10)
Accession_No	Varchar (15)
Phi	Int (3)
Sci	Int (3)
2nd_structure	Text
Inserted_By	Int (11)

The 3rd Structure table

One of the most difficult tables to configure was the 3rd Structure table. In the initial designs it would only hold 5 fields, but after much consideration and research it now holds 11 fields. This table started as a subset of attributes that were to be included in the 2nd Structure table but we soon realized that more attributes were needed. The major difficulty was to import all the data from separate files into the table. The structure files are linked to the PDB file that holds all the accession numbers and the Microbe information and importing all that data in the database while keeping that cross reference was a challenge. The initial design of the Peptide Structure table had a foreign key to the 3rd Structure table, but because of the import method and the 1-

N relationship to the Peptide table it was changed to just hold the foreign key of the Peptide table. If needed to link the 3rd_Structure table with the Peptide_Structure we can do so by querying through the Peptide table. As it was mentioned earlier, first the file with the accession numbers from the PDB file are imported, and then the separate files one by one. Currently the design is to have the 3rd Structure table having the two foreign keys from the Peptide table, those being the Accession Number and the Chain that ties the information from the files and the separate tables together. These two values provide unique identification of entries. The relationship between 3rd Structure and Peptide table is 1 – N and that is why creating a direct link is possible in this situation.

The importance of the Structure table is that it holds all the information about the three dimensional structure of a single, double or triple bonded protein molecule [14]. The first fields are the Amino Acid residue names in the first, and with their sequence number in the second field. The Atom numbers are represented in a third field, along with the Atom Names in a forth. Next are the X, Y and Z coordinates, represented in three different fields as real numbers, positive and negative. The final field before the Inserted_By attribute is Error or otherwise called the Temperature Factor.

Table structure:

Attribute:	Data type:
Unique_ID	Int (10)
Accession_No	Varchar (15)
AA_Names	Varchar (255)
Atoms	Int (3)
Atom_Names	Varchar (50)
X	Float (8,4)
Y	Float (8,4)
Z	Float (8,4)
Chain	Varchar (50)
Sequence_#_AA_Names	Int (5)
Error	Float (8,4)
Inserted_By	Int (11)

The Journal table

Three new tables were created in the design process and implementation of this database; the Journal table, the Deleted table, and the Notes table. The Journal table holds only 3 values. The Accession Number is the foreign key to the Peptide table, the Text is the Journal Link and reference to the scientific paper from which the Peptide was derived, and the Inserted_By field. In this case the Chain which consists of the candidate key from the Peptide table is not needed since the data file that holds the information has one Journal per Accession Number.

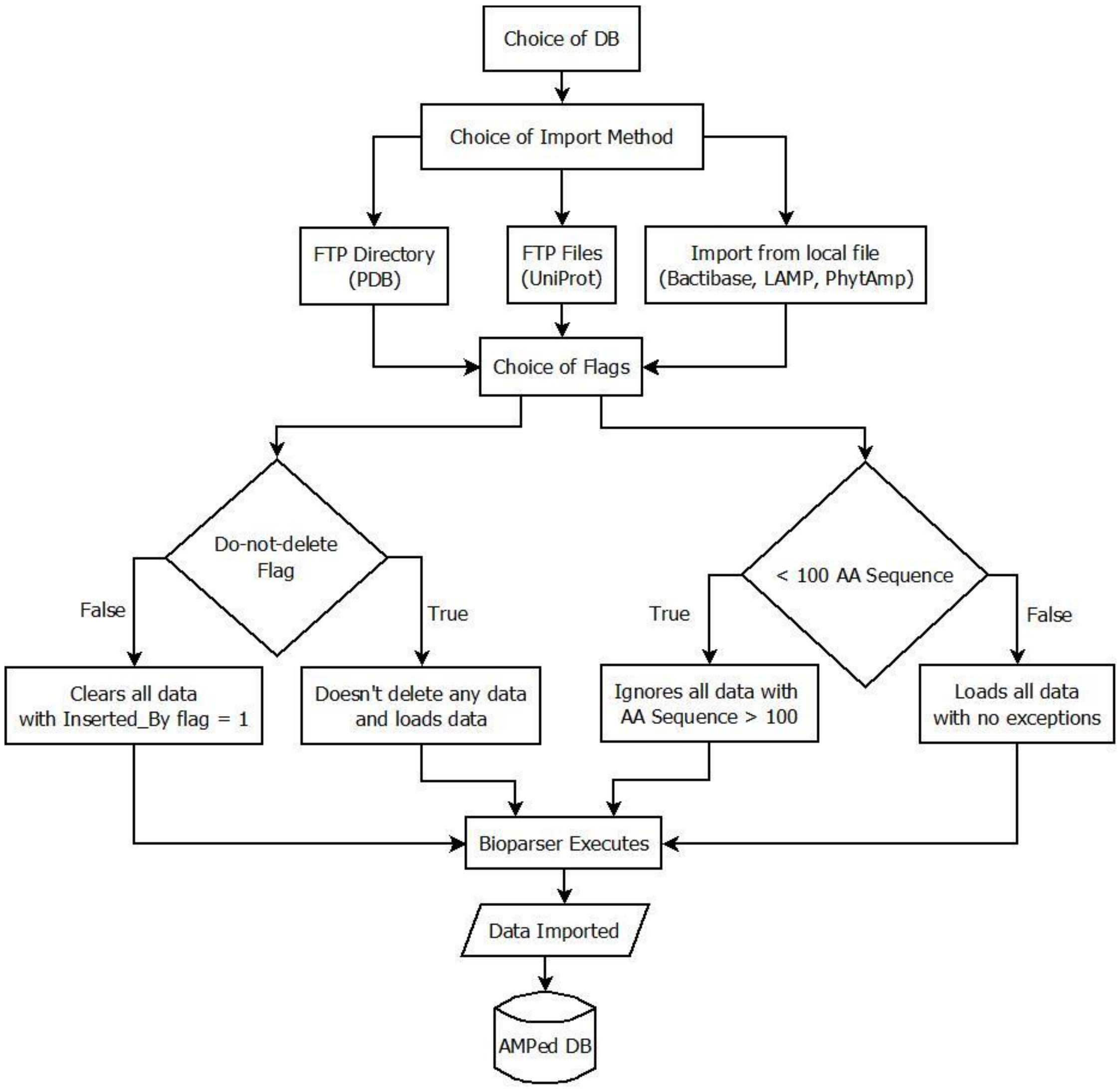
The next two tables support the automatic import of the data from the data files. The Deleted table holds the Accession Number and the Chain, and it acts as a placeholder for all the “deleted” data. The Notes table holds the same two values, but in addition it has a Notes field stored as text for any particular notes the team would like to add or calculations stored. This table stands by itself because when data gets deleted from the mass import process, any notes that were associated with that entry need to remain in the database. It uses the Accession_No and Chain as keys to always pull the notes plus the newly imported data.

Finally there is an additional table only placed as a reference, but it can be removed if not needed in the future. This table is called Inserted_By and it specifies the meaning of Inserted_By values that are used in other tables: 0 is for “Human” and 1 is “Automated”.

Table structure:

Attribute:	**Data type:**
Unique_ID	Int (11)
Accession_No	Varchar (15)
Text	Text
Inserted_By	Int (11)

d. Next Steps



Choice of DB
Choice of Import Method
FTP Directory (PDB)
FTP Files (UniProt)
Import from local file (Bactibase, LAMP, PhytAmp)
Choice of Flags
Do-not-delete Flag
False
True
< 100 AA Sequence
True
False
Clears all data with Inserted_By flag = 1
Doesn't delete any data and loads data
Ignores all data with AA Sequence > 100
Loads all data with no exceptions
Bioparser Executes
Data Imported
AMPed DB

Figure 7. Flowchart of the process to the database.

Currently we have only five databases loaded into AMPed. Future developers of Bioparser can extend it to parse more files with different formats. We have described how the data files are constructed, and how the tables and configured. Next we need to describe the import process that moves data from the various database files to our database.

The three smaller databases, BACTIBASE, LAMP, and PhytAMP, will be loaded from local files using the “one shot” mode. While importing values, in order not to delete the data, the flag “do-not-delete” is used from the user at the command prompt. The Bioparser tool is built in such a way that it will delete all data with the value of Inserted_By = 1. This process happens because in the import process the files that contain the data are not selectively imported but in bulk. The data that is imported set the Inserted_By flag with a ‘1’, and any human intervention will be marked as ‘0’, so the data will not be wiped. So in order not to lose the existing data in the database, one can set all the Inserted_By values of those databases to 0, or just set the “do-not-delete” flag of the Bioparser tool while importing. Because the three databases are not going to be updated on a monthly basis, then their Inserted_By values can be set to 0 to assure data integrity. In addition two out of three databases, BACTIBASE and PhytAMP, can have their structural data (data from the 3rd_Structure table) inserted in the meantime, so we do not want to have the previous data deleted.

Once these three databases are inserted, then the next step is to load the UniProt file. This file holds information from multiple tables, so while inserting the data in AMPed it matches the keys and the tables with all the corresponding data. For example, the name and the accession number will go into the Peptide table, but there is also information about the species of the organism, the associated microbes type and the journal reference. In real time it first stores data in the Peptide table and then it pulls the Accession Number and the Chain as the primary keys to cross reference the rest of the data across the Genome, Microbe, Journal and the two cross referencing tables which are Fight_Against and Coded_by.

Finally the PDB files are inserted in a two-step process. As we mentioned earlier, the pdb_seqres file holds the basic information that is stored in the Peptide table, so first that file needs to be imported. During the second phase, for each entry in the pdb_seqres file there is a single .ent file that corresponds to that entry. The Bioparser tool logs in the FTP folder and downloads all the .ent files that exist there. Next, it will go through each file one by one, and stores the data in the appropriate tables based on the Accession Number and the Chain. During this process the tool also takes into consideration the length of the amino acid. A sequence of 100 amino acid residues long or less will be used, and anything greater than that it will be discarded, because peptides are considered only the proteins containing anything less than 100 amino acid long. For the 3rd_structure and the Peptide_Structure table the tool does not have to query the Peptide table because it stores directly the Accession Number and the Chain. For information though that exist in the .ent file and corresponds to either the Microbe table or the Genome table, it first queries the Peptide table based on the primary key and then stores the information in the Genome table, and the Coded_by table, or the Microbe table and the Fight_Against table.

After all data has been inserted into the database, the next step is to create the correct queries in order to pull the data needed by the team. Data will not be deleted, but it will be stored in the Deleted table in order to be hidden from the front-end user. All queries will be checking for any entries stored in the Deleted table, but this data will not be shown. Dr. Martin and her lab group that will be maintaining the data might eventually decide to discard some peptides from the database, but for historic reasons and audit reasons the data will remain in the database just not shown to the web interface. For data that needs to be modified the Notes table has been created. Since most data will be deleted upon every update of the database, data that needs to be modified or altered will be annotated in the Notes table. A field that will hold all notes or comments based on the accession number and the key will be stored in that table. Moreover for data that needs to be "modified" the Inserted_By field will be set to 0, so that in the next update it will not be deleted; the same entry will re-imported. As noted earlier, the Inserted_By field indicates which entry was altered by a human, and which entry was taken directly from a data file. In addition a trigger will have to be created when a user manually updates an entry that will join all tables for that specific entry and flip the Inserted_By flag to 0 for manual annotation. When the next team creates the queries for the web interface, they will have to take into consideration the automatic versus manually annotated data. In order not to join data with the wrong information, the join queries will have to include the Inserted_By flag. Currently in the database the value that links most tables is the Accession_No and the Chain. If any entry was manually annotated then the Inserted_By flag would flip to 0. On the next update of the database, that entry would not be deleted, but the same one from the source file would be re-imported with the Inserted_By flag equal to 1. Since the Accession_No's will be the same for both entries, the only thing that would distinguish them across all tables would be the Inserted_By flag, and that is why it would need to be included in the join queries.

In the future it might be useful to create a table to keep a unique idenitifer for the AMPed database. For example, every database that gets imported has a unique identifier like P4098 or 3FR9. Currently because of the bulk imports, the database cannot keep a constant identifier. For that reason an extra table that will not be affected by the deletion process and the import process will be needed. This table will hold only three values, the unique identifier being AMPed's accession number, the accession numbers of the rest of the databases, and the chain to make them unique when talking for example for PDB files. This might make the querying of the tables a bit slower because of all the JOINs that will have to take place, but it will eliminate the risk of the accession number changing based on the deletion and re-import of all the data.

V. Parsing and Importing Data

In order for any loading of the data into the database there must be a process to first find the data files, download them, and then parse them. For that reason, a tool was developed by our team to handle each case for each different file type. The tool was built with the Go Language [20], a language built by Google and started as a stable release in March of 2011 [21]. Considering the history of the language it is fairly new compared to C, C++, or Python, which have been around for many years.

The final release, and the one in which this program was coded, is go1.2, released in December of 2013. This language was chosen after research on existing parsing languages, because of its many advantages. “Programs in Go are constructed from packages, whose properties allow efficient management of dependencies.” [22]. Dependencies in Go language are the links between the packages and their target, being the main program. Go programs compile very fast, their execution time is short, and code can be written very efficiently (in very few lines of code). It has some disadvantages like any other programming language; it is not object oriented in the traditional way, and it is still a “young” language. A language that has only been around for 1-2 years could mean that it could change more often, so that could require code changes on the side of the developer, or updates to keep original functionality. Since its first release, it has evolved and it will continue to evolve in ways that many believe will likely improve the language.

The following section will discuss the download method, the parsing method and the import method of the Bioparser tool.

The download method

We created this tool to automatically download and import data into the database. There a few ways to use the Bioparser tool: to download all similar files from a specific FTP location, to download a single file, and finally to load a file from a local repository.

The tool is run from the command prompt or terminal. There is a help section that shows the options available when running it, just by typing *"bioparser –help,"* The help section also show what flags need to be set in order to execute the Bioparser. In order to run the download option for multiple files the following command has to be typed in the terminal "*bioparser –ftp-dirs "<URL_will_be_entered_here>"* ". At this point the Bioparser connects to the FTP server, authenticates itself, and then starts downloading the files one by one, and while it is downloading it will start parsing the files.

The next option we can choose is downloading from FTP single files. The tool can be run for that option by typing in the command line *"bioparser –ftp-files "<URL_will_be_entered_here >" "*. Multiple URLs can be entered at this point by separating them with a space. The example that falls in this category is the uniprot_dat file and the pdb_seqres file.

And the third option, in which the Bioparser can be run, is to load single files from the local repository. This method is used as a fail safety mechanism and for files that need to be added on top of the re-import method. In case the download does complete, but the socket of the MySQL fails, or the parser fails to read files, then they don't have to be re-downloaded but just parsed and imported into the database. In order to run the tool with this method we can type *"bioparser –file "<path_of_file_in_local_repository>"".*

The parsing method –ftp-dirs

The Go language utilizes available packages and its functions for specific actions. For example, the "flag" package is used to set flags in the main program. Then, the function flag.Parse is used to parse the command line to read all available flags so they can be used later in the program. It will pick up the "—ftp-dirs" flag and accordingly run the ModeFTP() function.

Parts of the main code:

Below we see the packages being imported in the code in the very beginning in order to be used further down.

```
import (
        "code.google.com/p/ftp4go"
        "database/sql"
        "errors"
        "flag"
        "fmt"
```

```
		_ "github.com/go-sql-driver/mysql"
		"github.com/russross/meddler"
		"os"
		"strings"
		"sync"
)
```

After the packages are loaded the input from the command line is read by the following code.

```
var opt_ftp_dirs = flag.String("ftp-dirs", "", "the directories to pull. use spaces to separate each directory")
```

At this point the main() program will open the connection to mysql and then determine If we need to run ModeFTP() or ModeSingleFile().

```
flag.Parse()

   {
      tdb, err := sql.Open("mysql", *opt_db_user+":"+*opt_db_pass+"@"+*opt_db_addr+"/"+*opt_db_name)
      if err != nil {
         panic(err)
      }
      err = tdb.Ping()
      if err != nil {
         panic(err)
      }
      db = tdb
      tx, err = db.Begin()
      if err != nil {
         panic(err)
       }


if len(*opt_file_name) > 0 {

			fmt.Printf("Running in oneshot mode\n")

			ModeSingleFile()

		} else if len(*opt_ftp_files) > 0 || len(*opt_ftp_dirs) > 0{

			fmt.Printf("Running in FTP mode\n")

			ModeFTP()

		} else {

			fmt.Printf("Must provide --file or (--ftp-files or --ftp-dirs)\n");

		}
```

Initially the ModeFTP() function will set the temp directory where the files will be downloaded and it is currently set to “tmp/bioparser”, but this can be set with the “-tmp” flag in the command line. Afterwards the function creates a query for deleting all data with the Inserted_By value of 1, if the “-do-not-delete” flag is set. This functionality is the backbone of the re-import of all data files, since it clears all the tables and it re-imports all updated data again. Continuing, the “ftp4go” package is used to set all the parameters that will be passed in order to connect to the FTP server that was given in the command line. This package can handle the FTP connections, so its parameters will be the address and the login name for that connection. It will additionally set the parameters for downloading a file and then loop around each file to download them locally. Once it has listed all the files in the current directory, it will call the DetermineParserByFilename() function to determine which parsing method it will use based on the file name.

Part of code of ModeFTP() still within the main.go file:

```
//SplitN slices directories into substrings separated by " " and returns a slice of the substrings between
those separators.
        dirs := strings.SplitN(*opt_ftp_dirs, " ", -1)
        for _, dir := range dirs {
           parts := strings.SplitN(dir, "/", 2)
           if len(parts) != 2 {
              panic("missing path for ftp file " + dir)
           }
           s, err := get_server(parts[0])
           if err != nil {
              panic("Got error setting up FTP: " + err.Error())
           }
           _, err = s.Cwd(parts[1])
           if err != nil {
              panic("Got error changing dir: " + err.Error())
           }
           fmt.Printf("Retrieving file list...\n")
```

```
files, err := s.Nlst()
if err != nil {
    panic("Error listing files")
}
fmt.Printf("LIST: %v\n", files)
for _, file := range files {
    if file == "." || file == ".." {
        continue
    }
        p := DetermineParserByFilename(file)
```

Now that the files has been loaded and it has decided in which mode it will run, it needs to determine which parser out of the five it needs to run. The code below does that for us.

```
func DetermineParserByFilename(name string) Parser {

        lc := strings.ToLower(name)

        if strings.Contains(lc, "ent") && !strings.Contains(lc, "phytamp") {

                if strings.Contains(lc, "gz") {

                        return &ParserPDBStructure{Gz: true}

                }

                return &ParserPDBStructure{Gz: false}

        } else if strings.Contains(lc, "uniprot") {

                if strings.Contains(lc, "gz") {

                        return &ParserUniprot{Gz: true}

                }

                return &ParserUniprot{Gz: false}

        } else if strings.Contains(lc, "pdb") {

                return &ParserPDB{}

        } else if strings.Contains(lc, "fasta") || strings.Contains(lc, "lamp") || strings.Contains(lc,
"phytamp") || strings.Contains(lc, "bactibase") {

                return &ParserFasta{}

        }
        return nil
}
```

So far, the only directory with multiple files that we set up to be downloaded continuously and be parsed automatically is the pdb structure files directory. The function at this point checks the names of the files and their extensions to see which parser the file will be passed on, to go through the information and extract it. For the pdb structure files the ParserPDBStructure is called, with the file passed as a parameter.

The ParserPDBStructure function initially checks if the file is compressed in a gzip or not and then calls accordingly the appropriate reader. For every file it scans every line in the file, and removes any leading spaces, semicolons, tabs, or returns. For every line that it scans it selects a case from the different headers that exist in the beginning of the line and performs an action. There are different headers that the parser needs to deal with. The first one is the “HEADER” where it extracts the accession number from. Then there is the “COMPND” where there is information if the peptide was designed or not, in a lab. Following there is the “SOURCE” where it gives information about the species of the peptide, and the species of the microbe. Since the information that needs to be passed is between the Peptide table, the Genome table, and the Microbe table, an UPDATE statement needs to be run. It uses the accession number as the foreign key from the cross reference tables, Coded_by and Fight_Against and updates the Microbe and Genome table, everywhere that finds the corresponding information.

The following headers that will get read in sequence in the ParserPDBStructure function are the “JRNL”, “HELIX”, “SHEET”, “ATOM”. For the “JRNL” it reads all the text and joins it together in a single string. The same thing happens for the “HELIX” and “SHEET” headers. For the “ATOM” header it will read each line as a separate entry in the 3rd_Structure table and insert it based on the accession number. In this case first it goes through the entire line and makes sure that it is 11 fields, and then separates them out by spaces and stores them in variables in order to send them in the database as an INSERT statement.

Some problems did occur with the parsing of the data, and that was why the structure of the tables has been changed. In order to be able to insert the data or update directly from the file, there had to be as few cross reference tables as possible. That is because the Bioparser tool uses loops to do an insert into the first table and then a post-insert loop to grab the unique id and cross reference it. Since the relationships of all the tables is not M-N but 1-N, then the primary key can be used as a foreign key directly in the other table without a cross reference between them. Also after we did research on the data files, there was realization that some fields had to have their own tables, just like the journal reference. This also happened because there is only one journal reference for multiple peptides, so in order to eliminate empty data rows in the Peptide table it was removed as its own. From the pdb structure file there are entries that need to be parsed more than once in order to capture all the values like the atom fields or in some cases the source. But because the journal reference only appears once for every accession number and it is the same even for all chains it had to be moved to its own table for avoidance of redundant data.

The parsing method –ftp-files

The second method for using the Bioparser tool is to use the "–ftp-files" option. With this option the tool downloads, from FTP servers, single files that can be specified by their URLs. The types of files that are parsed in this way are currently the uniprot_dat, and the pdb_seqres file. These two files have a completely different format so for that reason different parsers were developed.

Similar to what was described in the previous section the process the tool follows within the code in order to parse is the same. The function DetermineParserByFilename() is a method that tries to call the correct parser based on the name of the file. Starting off with the uniprot_dat file, it has a similar structure in concept like the PDB structure files that were reviewed previously. It uses headers to determine the information being passed in that line in the document. The function parsing this data file is called ParserUniprot().

ParserUniprot() checks if the file downloaded is zipped or not and runs the correct reader. Once the check is complete, it calls a “go” routine that reads all the lines in the file until it reaches the end of the loop, and that signifies one single record in the database. The parser goes through a switch statement that tries to locate all the headers, and extract the information from the lines and send them back to the channel for import.

The headers that will be parsed will be “AC” which is the accession number, “DE” being the name of the peptide, “OS” the organism source, “OH” organism host, “RX” the journal reference, and “SQ” which is the sequence.

The second file that will be downloaded under the same method to be updated in the database is the pdb_seqres file, and that holds the sequences and the names mostly of the PDB entries. The function that handles this kind of file format is the PDBParser(). After it has gone through the same process until the function DetermineParserByFilename(), then it calls the PDBParser() based on the filename of “pdb”.

This function will start a reader, to read through the lines of the file. The lines of the file are then passed to another function called FastaRead(), which handles all files that are based on FASTA format. The function will read each line and return an entry every time it encounters the ‘>’ operator. All FASTA format files begin an entry with the ‘>’ operator, so it makes it easier to call a single function that will handle all similar files. For each line there is handler that deals with the different information.

The first line always holds the accession number and name of the peptide, and the second line the amino acid sequence.

The parsing method –file

Last option that the Bioparser tool offers is the '—file' method. In this case the files are loaded in the database, in a one shot mode, or backup mode. When inserting data files in AMPed in this way, it will not delete the existing data because it is built as a safety mechanism. Also for many files that will be downloaded manually and need to be imported only once, this method can be used. In this case there exist 3 files that will be imported in this way, the BACTIBASE, the LAMP, and the PhytAMP files.

Depending on the name of the files, the function DetermineParserByFilename() will call the appropriate function to handle the parsing. The three files that are going to be uploaded once, are all in FASTA format and also pipe '|' delimited. That makes it easier to create a function to handle the base logic of the parsing, and with if-then statements split the files with different parameters.

All three files have two lines for each individual entry, so the second line will always be the amino acid sequence. The first parameter of the first line is the accession number, and the second parameter is the name of the peptide. For the PhytAMP file those are the only parameters existing to be parsed. The BACTIBASE has an extra parameter in the first line, which are any notes about the sequence or the peptide. And the LAMP file has in total of 6 parameters. The third parameter is the species of the genome, the fourth is the source of the peptide, and the fifth is the type of source: natural, experimental, or designed. The final parameter is the type of microbe the peptide is fighting. If the parser cannot handle any of the files above it will send back to the channel an error message stating in which line is the error, and the information in the line.

The import method

After the download of the files has taken place and the parsing has completed, the last part is importing the data in AMPed. That is handled mainly by the CommandHandler() function which uses initially a for loop to go through all the information coming through from the parser, and then uses a switch statement to decide how they will be sent as queries to the database. Below the switch statement where all this takes place is shown, and also the SQL queries that are prepared with a function. Before the CommandHandler() runs, the HandleFiles() function runs in order to initialize a parser and start a thread in order to pass the output data into the command handler.

```
func HandleFiles(fchan chan string) chan bool {
        cmds := make(chan interface{})

        end := CommandHandler(cmds)

        go func() {
                defer close(cmds)
                wg := sync.WaitGroup{}
                for file := range fchan {
                        fd, err := os.Open(file)
                        if err != nil {
                                panic("Couldn't open file in HandleFiles: " + err.Error())
                        }
                        p := DetermineParserByFilename(file)
                        tcmds := p.Parse(fd)

                        wg.Add(1)

                        go func() {

                                defer wg.Done()

                                for c := range tcmds {

                                        switch ce := c.(type) {

                                        case *NonFatalParserError:

                                                ce.FileName = file

                                        }

                                        cmds <- c

                                }
```

Part of code of CommandHandler():

```
switch t := obj.(type) {
      case ToSql:
        //fmt.Printf("Preinserting: %s = %v\n", t.TableName(), t)
        if pi, ok := obj.(ToSqlPreInsert); ok {
          pi.PreInsert()
        }
        //start := time.Now();
        //fmt.Printf("Inserting: %s = %v\n", t.TableName(), t)
        err := meddler.Insert(tx, t.TableName(), t)
        //duration := time.Since(start);
        if err != nil {
          panic(err)
        }
        //println("TOOK: ", duration);
        if pi, ok := obj.(ToSqlPostInsert); ok {
          if post_insert_not_started {
            post_insert_retchan = CommandHandler(post_insert_queue)
            post_insert_not_started = false
          }
          pi.PostInsert(post_insert_queue)
        }
      case *RawSQL:
        _, err := tx.Exec(t.SQL, t.Args...)
        if err != nil {
          println(t.SQL)
          panic(err)
        }
      case *NonFatalParserError:
        t.FileName = *opt_file_name
        fmt.Printf("Got Parse Error: %v\n", t)
        fmt.Fprintf(logfd, "Got Parse Error: %v\n", t)
```

Part of the Tables code:

```
type Peptide struct {
   Id         int    `meddler:"Unique_ID,pk"`
   AccessionNo string `meddler:"Accession_No,zeroisnull"`
   Chain      string `meddler:"Chain"`

   Name       string `meddler:"Name"`
   MolWeight  string `meddler:"Mol_Weight,zeroisnull"`
   AASequence string `meddler:"AA_Sequence"`
   LengthSeq  int    `meddler:"Length_seq"`
   Hydro      string `meddler:"hydro"`
   Notes      string `meddler:"Notes"`
   InsertedBy int    `meddler:"Inserted_By"`
}
```

The tables are defined in order to be used as SQL queries within the CommandHandler() function. The function will use two main cases in order to send the queries in the database, either as an insert statement, or as an update statement that is used to enter additional information to pre-existing entries. Within the first case, the package ‘meddler’ [23] is used to execute the INSERT statements from the pre-defined tables to the database. In the second case the RawSQL struct is called which executes an UPDATE statement, after it has inserted the primary keys to the cross reference tables, to insert additional information that has been parsed. In addition there were three more cases added to the switch statement for any errors that might come up. The case in which there is a non-fatal error is shown above as part of the rest of the code. The first error would be a non-fatal parser error, which doesn’t stop the parser from continuing, but prints the error on the screen. The second error would be a fatal error were the parser would completely stop and exit because file handling. The third error is simply for anything that cannot be handled and cannot be expected.

The main difficulty in the import process was the order of the INSERT and UPDATE statements, because of the way the files store their data, and the way the database is built. Because most of the files will be using similar techniques for uploading data, there was one interface used to capture that general concept, and for files that needed to be updated a second interface would be used to capture the information already stored to populate the cross reference tables. Also another problem that rose was the blocking of the thread that would be pulling the data in order for all insertion completed, and then the update would happen. That would slow down the process though by a significant amount of time, so a decision was made to do a post insert statement right before the case exited, so it would insert the information in the cross reference table. That is also shown previously in the part of the code from the CommandHandler() function.

Parts for Improvement

The database and the parsing tool have been developed by our team to handle the most common and widely searchable scenarios that scientists come across. There is a lot more information about peptides that could be utilized that is currently missing from the database, if there was ever a need to expand AMPed. The files that have additional information so far, that are not used, are the PDB Structure files and the Uniprot file. To be able to handle the extra information it would require modification of the parsing tool.

In addition the parsing tool currently handles the parsing of the third structure information based on the number of fields existing on its line in the file, and the number of empty spaces. The correct way to develop the parser would be to be count the number of characters from the beginning of the line and extract the information based on the character position. When developing the code for that functionality this assumption that every field will have a space before the next field was taken. As it was later discovered there every field has specific characters assigned to it, and they are all used according to the data. The parser could not read the two separate values because there was no space and the was causing a non-fatal error. We finally fixed this issue as a requirement towards the end of the thesis since the order of the atoms was very important for Dr. Martin and her lab.

As said earlier there is also more room for more sources to be added to the list of the data files currently stored in the warehouse. If additional files are based on FASTA format the implementation would be rather easy, compared to a custom made file format that would require more in depth analysis and development of custom extraction of data.

Another part that could require more research is the database's unique accession number. As of today the primary keys that exist in AMPed have no meaning because of the way the re-import method happens. The primary keys which are just numbers keep increasing with every update operation. One possible solution to this issue would be to create a separate table that will store every unique accession number and chain from the imported data. That table will only hold three values, AMPed's unique accession number, the imported file's accession number, and the chain. In that way with a simple JOIN between tables every entry in the database can have its own static accession number. That might slow down the querying speeds of the database, but it will ensure a static identifier for all entries. Again as a requirement for the completion of this thesis that table was developed and populated. It currently holds all of the accession numbers from the Peptide table, and a new accession number for our AMPed database which has a format of AMPxxxxxx, where 'x' is a number sequence incremented sequentially.

VI. Future Work

This project has many more parts than this thesis, and some are yet to be completed. The major parts of this thesis are the initial design of the database, the implementation of the database, the tools that will download, parse, and import the data, the web interface for user interaction, and finally calculation tools that will have to pull from the data in order to produce an outcome. The first three parts have been completed with the completion of this thesis by our team. The next parts are to be completed either by the AMPed team, or by future graduate students that will work on the continuation of this project.

Additional features of the web interface still need to be developed. First of all the interface has to be written in a language that supports back end interaction with a MySQL database. Secondly it has to be open source, and easy to interact with other open source scripting tools or similar alternative. Based on these requirements, the suggested language is be PHP with some HTML and CSS for better graphic layouts. Using PHP, queries can be written to directly use INSERT, UPDATE, or even DELETE statements, based on the user actions or on the predefined actions available.

The scripting language for the calculation tools can be from a variety of options, such as Python, Ruby, Ruby on Rails, or Perl. One of these calculation tools based on the Chou-Fasman algorithm has already been developed by Greg Gardner, an undergraduate student. It has not yet been implemented with the database because the web interface is not developed yet. The language of choice in this case was

Python but that doesn’t restrain the rest of the tools to this language. Any other additions or features of this project will be discussed further in the future.

VII. Conclusions

The AMPed project is a collective effort over the past few years to create a uniform and complete collection of antimicrobial peptides by storing them in a database warehouse. The database design was developed by our team in such a way that all of the requirements from the rest of the sources will be stored uniformly into one. Although the design as it stands now is complete, that does not mean that it is not extensible and cannot be modified for future needs. The research on biological data will continue to expand, and this database is meant to keep up with that improvement.

Based on our goals from the introductions we have managed to accomplish all of them in specific ways. The first objective was to find a way to download the data. The Bioparser tool accommodates that need by offering two different ways of downloading the data, either by downloading single files or entire ftp directories. The second objective was a way to import the data, and by utilizing again the Bioparser tool we accomplished that. This tool will download and import data, with the two options mentioned, or it can simply import from already downloaded files. After downloading and importing the data for the first time, we had to fulfill the need of regular updates. That was met by inserting a flag in each table of the database, called Inserted_By, and every time data was manually annotated it will flip to 0 (human intervention). Alongside with the Inserted_By flag, we have made the Bioparser tool to take into consideration that flag, and wipe out everything that is set to 1 (automatic import), in order to re-import all data, but at the same time, preserve all annotations to the old data. In addition an extra field for further notes has been created in the Notes table. Our fourth goal was to find the quickest way to import that data, and that is why the choice of Go Language was made. It can import a single file of 2.8GB in under 3 minutes. When importing entire directories it can slow down a bit based on the amount of files it has to parse through. Finally the normalization of the data was met by constant revisions of the database, and continuous meetings with the team. A final E-R design has been made and agreed on by all members of the team and it can be seen in Appendix A.

The web interface that will be built will give the researchers the opportunity to view all the data from different databases collected in a one location. Moreover there will be a possibility of entering new data through the use of the interface and using the calculation tools to analyze the experimental data. This project has had many positive and negative results in the process. In the end with the proper collaboration and communication between computer scientists and biologists, a result that will benefit different people has been reached. In the near future this opens up new doors for experimentation and research while utilizing the knowledge of both parties.

VIII. Appendix A

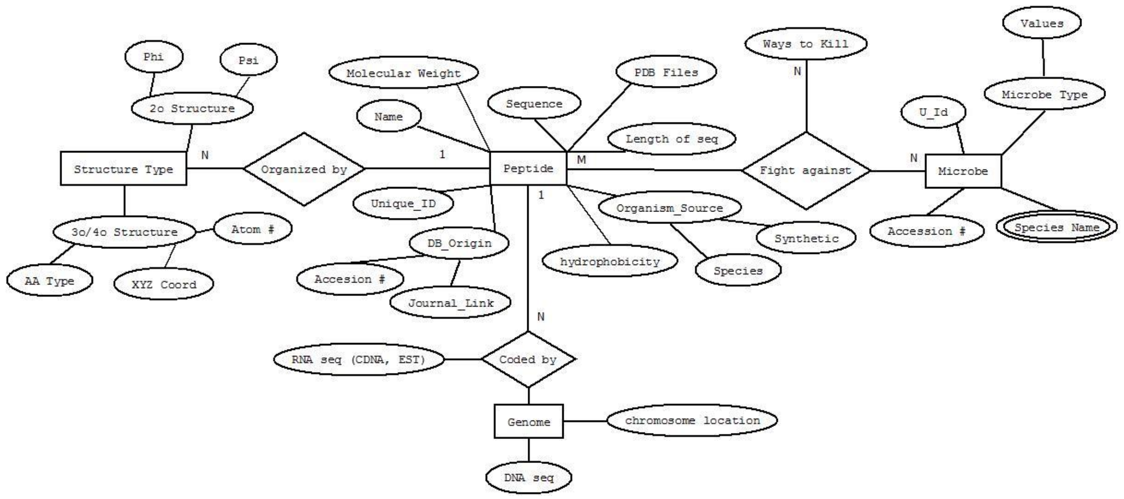


Figure 8. Complete E-R Diagram.

IX. Appendix B

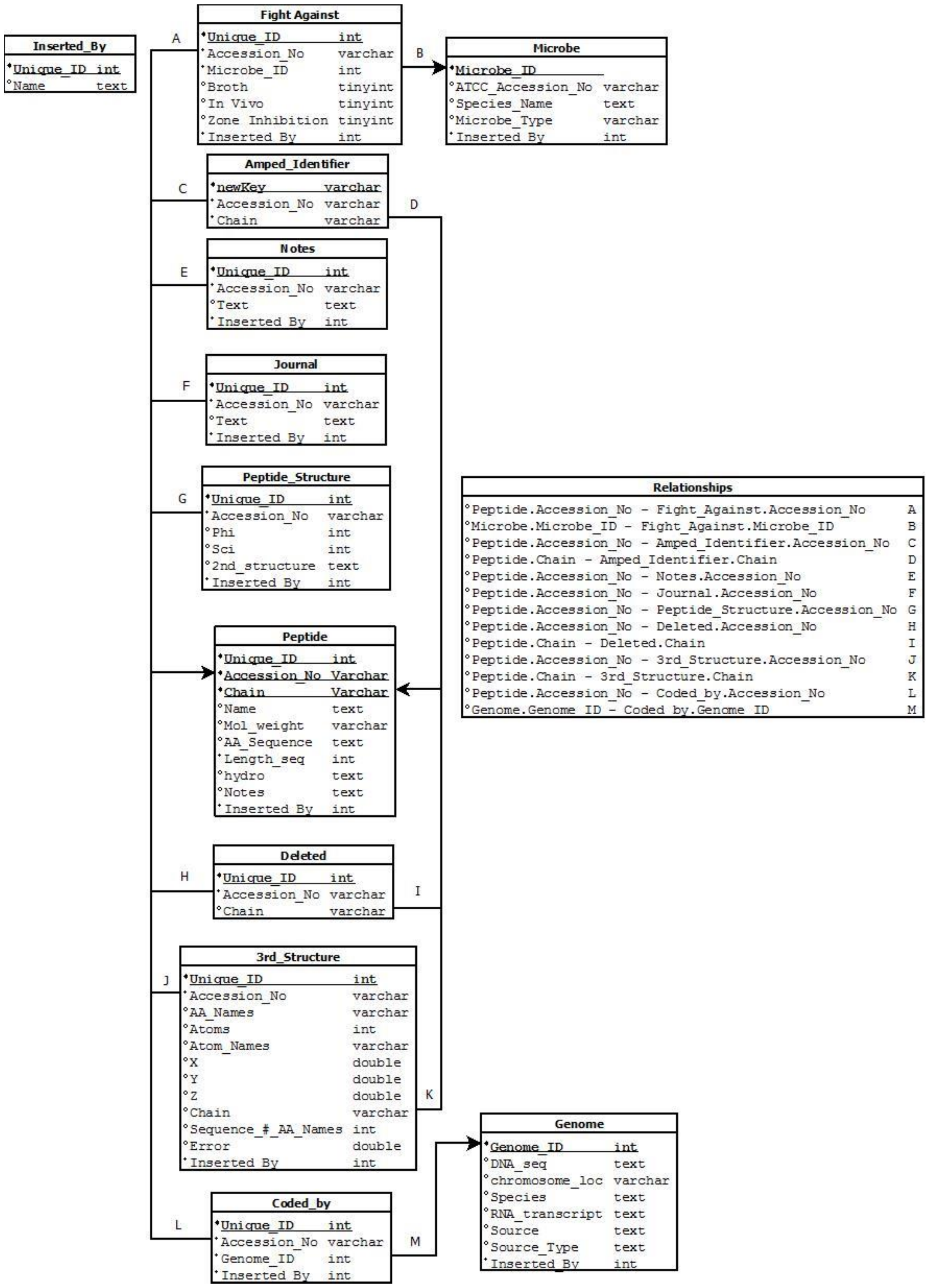


Figure 9. Complete structure of the tables in AMPed.

X. Bibliography

1. David Ryder, M.Sc. Biochemistry Thesis, URI, Kingston, RI, USA, 2003.
2. Last accessed: 04/15/2014, URL: http://www.rcsb.org/pdb/home/home.do, RCSB Protein Data Bank, RCSB-Rutgers Department of Chemistry and Chemical Biology, RCSB-SDSC San Diego Supercomputer Center and Skaggs School of Pharmacy and Pharmaceutical Sciences.
3. Last accessed: 04/01/2014, URL: http://aps.unmc.edu/AP/main.php, The Antimicrobial Peptide Database, Guangshun Wang, Nebraska, USA.
4. Last accessed: 04/01/2014, URL: http://yadamp.unisa.it/, YADAMP – Yet another database of antimicrobial peptides, Soft Matter Group, University of Salerno, Italy.
5. Last accessed: 01/15/2014, URL: http://apps.sanbi.ac.za/dampd/index.php, DAMPD – Dragon Antimicrobial Peptide Database, Professor Vlad Bajic, Dr. Sundararajan V.S., OrionCell, South African National Bioinformatics Institute.
6. Last accessed: 04/15/2014, URL: http://phytamp.pfba-lab-tun.org/main.php, PhytAmp – A database dedicated to plant antimicrobial peptides, Hammami Riadh, Ben Hamida Jeannette, Vergoten Gerard, Fliss Ismail, Institute of Applied Biological Sciences Tunis, Tunisia.
7. Last accessed: 09/15/2014, URL: http://biotechlab.fudan.edu.cn/database/lamp/, LAMP – A database linking antimicrobial peptide, Dr. Qingshan Huang, Xiaowei Zhao, Jinjiang Huang,

Hongyu Wu, Dr. Hairong Lu, Guodong Li, School of Life Sciences, Fudan University, Shanghai, China.

8. Last accessed: 8/30/2014, URL: http://www.uniprot.org, The UniProt Consortium, Reorganizing the protein space at the Universal Protein Resource (UniProt), Nucleic Acids Res. 40: D71-D75 (2012).
9. Last accessed: 08/30/2014. URL: http://bactibase.pfba-lab-tun.org/main.php, "BACTIBASE – database dedicated to bacteriocins", Functional Proteomics & Alimentary Bio-preservation Research Unit, Institute of Applied Biological Sciences Tunis (ISSBAT), Tunisia.
10. Last accessed: 09/30/2014. URL: http://www.ncbi.nlm.nih.gov/blast/blastcgihelp.shtml, National Center for Biotechnology Information, US. National Library of Medicine, USA
11. Amos Bairoch, Rolf Apweiler, Cathy H. Wu. "User Manual". UniProt Knowledgebase.
12. Last accessed: 09/30/2014. URL: http://filext.com/file-extension/ENT, ".ENT File", FilEXT – a free online resource by Uniblue, Uniblue Systems Ltd.
13. Last accessed: 09/12/2014. URL: http://goldbook.iupac.org/A00279.html, IUPAC. Compendium of Chemical Terminology, 2nd ed. (the "Gold Book"). Compiled by A. D. McNaught and A. Wilkinson. Blackwell Scientific Publications, Oxford (1997).

14. Last accessed 09/30/2014. URL: http://www.rpi.edu/dept/bcbp/molbiochem/MBWeb/mb1/part2/protein.htm, “Basic Concepts of Protein Structure”, Joyce J. Diwan, 2003.

15. Last accessed 09/30/2014. URL: http://www.nlm.nih.gov/medlineplus/ency/article/002327.htm, “Chromosome”, U.S. National Library of Medicine, National Institutes of Health.

16. Last accessed 09/30/2014. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/organic/transcription.html, “DNA to RNA Transcription”, Karp, Gerald, Cell and Molecular Biology, 5th Ed., Wiley, 2008.

17. Last accessed 09/30/2014. URL: http://mpkb.org/home/patients/assessing_literature/in_vitro_studies, “Differences between in vitro, in vivo, and in silico studies”, The Marshall Protocol Knowledge Base, Autoimmunity Research Foundation.

18. Last accessed 09/30/2014. URL: http://www.eucast.org/fileadmin/src/media/PDFs/4ESCMID_Library/3Publications/EUCAST_Documents/Publications/E_Def_1_2_03_2000.pdf, “Terminology relating to methods for the determination of susceptibility of bacteria to antimicrobial agents”, May 2000, EUCAST, ESCMID.

19. Last accessed 09/30/2014. URL: http://www.austincc.edu/emeyerth/tertiary.htm, “Proteins: Tertiary and Quaternary Structures”, Dr. Edward Meyertholen, Austin Community College.

20. Last accessed 10/25/2014. URL: http://golang.org/doc/effective_go.html, "Effective GO", Google.

21. Last accessed 10/25/2014. URL: https://blog.golang.org/go-becomes-more-stable, "Go becomes more stable", Andrew Gerrand, Google.

22. Last accessed 10/25/2014. URL: https://golang.org/ref/spec, "The Go Programming Language Specification", Google.

23. Last accessed 10/25/2014. URL: http://godoc.org/github.com/russross/meddler, "package meddler", 2013 Russ Ross.

[ii] The Chou-Fasman algorithm is very specific to biologists, and its usage is to find the alpha helices, bet sheets and turns based on an amino acid sequence.