Spring 2021

A Deductive Database for Knot Colourings

Dong Hyun Han
Bard College

A Deductive Database for Knot Colourings

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Dong Hyun Han

Annandale-on-Hudson, New York
May, 2021

Abstract

This work constitutes progress toward the development of a knowledge base for braids, knots, and their colourings. The main result of this development is the creation of a logical model for storing data pertaining to braids, two-dimensional projections of three dimensional knots, finite quandles, and colorings of braids and knots by quandles. It uses the Entity Relationship data reference model as its starting point and makes the original design there. In addition, it includes a conversion of the Entity Relationship Diagram (ERD) to SQL queries that define tables corresponding to the ERD entity sets. Finally this work demonstrates how to populate the database on a given set of data in the input format for the Color My Knot (CMK) application by McGrail, Nguyen, and Granda.

Contents

Dedication

To my dad, my mum, my sister, and most importantly, my dog Momo.
I love you guys so much.

Acknowledgments

I would like to thank my senior project advisor Professor Robert McGrail for his constant support.

I would also like to thank all my friends back in Korea, especially the Dream church community, for their belief in me.

Being on the Bard Men's Volleyball team in the past four years has been phenomenal and will be one of the best decisions I made at Bard.

Finally, I would like to thank all of my friends who supported me throughout my four years at Bard.

1
Introduction

Databases are great tools for managing a great deal of information over time [6]. Ullman [6] describes their primary goals as to storing large amount of data and serving information the users want. In our case, we use a relational model to look specifically at stored information about knots, braids, and quandles to see which entities are relevant in certain cases and how each entity and attributes are related to each other. There is a collection of knots on the web called the Knot Atlas [1]. The information on the Knot Atlas [1] stores all of the knots with their properties and how they look like. The Knot Atlas, however, is not a relational model in a sense which the information given is discrete from each other. The Knot Atlas [1] is the biggest and the only collection of data about knots that is public. This paper aims to take a step forward and to describe how each property of a knot, braid, and quandle, is ultimately associated with each other.

In Chapter 2, we provide background information on knots and braids on what they are and their properties, such as crossings, specifically all Reidemeister moves, and how they are important when describing ambient isotopy [3] in knots and braids. In Chapter 3, we describe what quandles [3] are and its properties along with their relationship to knots. In Chapter 4, we talk about our different standard input formats [2] for braids, knots, and quandles. The sections in this chapter try to describe how our inputs will look like

and what they mean. The fifth Chapter deals with relational databases. First, we give background information on what a relational database [6] is and how we use it to our advantage in finding what information is directly and indirectly related to each other. In Chapter 6, we describe how the given SQL DMLs [6] are used in the database and its eventual representation in the database.

2
Knots

2.1 Knot Presentation

A knot is a continuous embedding of circles in three-dimensional space R^3 [4]. In this paper, we will describe knots in three-dimensional Euclidean space and their two-dimensional projections [4]. In order for two knots to be the same, it must be proven that they must be projections of each other in some way in the provided space, which is known as homeomorphism. [4]. Suppose we have two knots that are almost identical to each other in the three-dimensional Euclidean space, and they are also a function of x. For two or more given knots to be the same, one knot must be a continuous deformation into the other knot. This is called ambient isotopy [3]. For example, the figure below is a projection of the trefoil knot. Notice that, this knot has three crossings. Crossings occur when two strands of a knot overlap each other, making an intersection at a specific location, and crossings are denoted with either the star(*) operator or the div(/) operator [4]. An apparently broken part of of a strand is called an arc. Here, we introduce the Alexander-Briggs notation of this knot, which is trefoil (3_1) [4]. For each knot presented in the Alexander-Briggs notation in n_k, n denotes the total number of crossings of the knot, and k denotes the variant of the knot of n crossings.

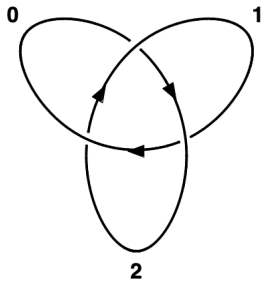

Figure 2.1.1. figure
The Trefoil Knot 3_1

2.2 Braid Presentation

Braids are another way to present knots. Braids are different from knots, in that they are presented with horizontal strands with crossings that happen vertically between two adjacent strands. The strands of the braids are implicitly aligned from left to right, and once the strand reaches the end, each of endings from the left side and the right side of the strands can be joined to form a continuous strand. This is called a link [3], and a link, ultimately, can represent a knot. Each crossing in the braid presentation is identified with a lower case sigma with a subscript and a superscript. The subscript of sigma denotes where the crossing happens on a strand. For example, σ_1 tells us that the crossing happens between the first and the second strand. The superscript of sigma can tell us two things [3]. First, the superscript of sigma tells us if the crossing σ^i at the ith index is inverse or not. Second, the superscript of sigma can tell us if there are multiple consecutive crossing at the same ith position. For example, in the braid presentation of the figure eight knot below, the first crossing, σ_1^1 identifies one non-inverse crossings that happens between the first strand and the second strand [3].

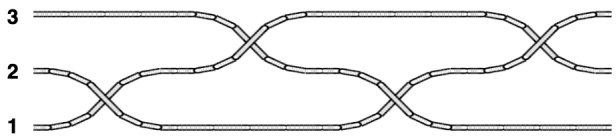

Figure 2.2.1. figure
Figure Eight Knot in Braid Presentation

2.3 Reidemeister Moves

Crossings in knots are what makes each individual knots unique. Ambient isotopy [3] and Reidemeister moves are related to each other, because we can prove that two similar knots in the same three-dimensional Euclidean space are the same by using the Reidemeister moves. There are three different types of Reidemeister moves we can perform on a given knot, which are Reidemeister move type I (idempotence), Reidemeister move type II (right-cancellation), and Reidemeister move type III (right self-distributivity). These move sets are defined visually in the figure below [5].

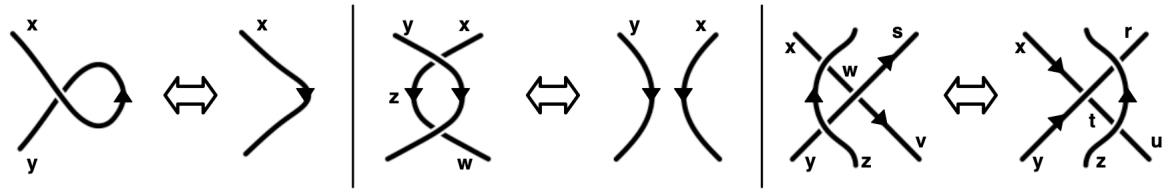

Figure 2.3.1. figure
Reidemeister Moves Types I, II, and III

3
Quandles and Colouring

3.1 Quandles

Quandles are sets of binary operations, star(*) and div(/), that satisfy axioms analogous to the Reidemeister moves utilised to handle knots. Let us define each of the quandle axioms below:

Reidemeister move type I: $X * X = Y = X$
Reidemeister move type II: $(X * Y) / Y = X$ and $(X / Y) * Y = X$
Reidemeister move type III: $(X * Y) * Z = (X * Z) * (Y * Z)$

Using the Reidemeister moves defined above as quandle axioms, we are able to convert the information of a knot diagrams onto a finite quandle, and a finite quandle is a n x n quandle that is used to of knots in the three-dimensional Euclidean space and generalised them in such ways, we can generalise them for all knots.

The labels on figure 3.1.1 constitute a colouring of the Trefoil knot (3_1) by the quandle of table 3.1.1. There are in total three crossings in the figure, and the crossing arithmetic we can derive from the knot quandle match the three crossings in the figure, such as $0 * 1 = 2$,

$1 * 2 = 0$, and $2 * 0 = 1$.

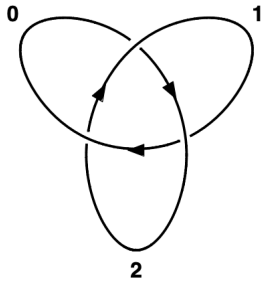

Figure 3.1.1. figure
The Trefoil Knot 3_1

*	0	1	2
0	0	2	1
1	2	1	0
2	1	0	2

Table 3.1.1. quandle of trefoil (3_1) knot

3.2 Colourings

Each colouring is labelled with a number in the row and column indices as noted in the quandle. Using this notion, we are able to label the colour of each arc and determine where the crossing happens in a given knot. Colouring of a knot is very important because they act as solutions to the quandle axioms, done by Reidemeister moves [3], which assign elements to appropriate crossovers within the quandle table. The colouring of a knot could be trivial, which means that there is only one colour due to the quandle axiom of idempotence, which is Reidemeister move type I [3]. This is reflected on the quandle as the diagonal values as seen in table 3.1.1. For knots with non-trivial colourings, we can determine that if a knot in a three-dimensional Euclidean space has a non-trivial colouring and another knot does not, these two knots are not the same. An example of a non-trivial colouring is the Trefoil knot (3_1) of figure 3.1.1.

4
Standard Input Format

In this chapter, we provide information about the standard input format for the necessary files for scanning and parsing for braids, knots, and quandles. The standard input format files are allknots.pres for knot presentations and CMK extension files for braid presentation files (braids_tr_7.cmk), and quandles (dihedrals.cmk) [3].

4.1 Standard Input Format for Knots

In this section, we provide what the standard input format for knots are and what each item means. The file that we will scan and parse for knots is called allknots.pres. This is a file of the standard input format that contains all of the existing knots, which looks like: The figure above is a sample line from allknots.pres, and this is the sequence for the Trefoil (3_1) knot. In order to scan and parse the necessary information, we need to be able to determine what each token means. The first word, presentation, is what denotes the start of a new knot sequence. The numbers 3 and 1 after the parenthesis is important,

presentation(3, 1, [-3, -1, -2]).

Figure 4.1.1. line of input from allknots.pres

because the first number after the parenthesis denotes the number of crossings within a given knot sequence, and the second number after the parenthesis denotes its variant. For knots with 3 crossings, there exists only one variant, but for knots with 10 crossings, there are more than 100 variants [1], so this is important when keeping track of each variation of knots with the same number of crossings. The numbers in square braces are the actual crossings. These numbers determine how the knot is formed. The important part about these numbers is whether they are positive or negative. The sign of the numbers denote the direction of the strand that crosses over another. The numbers in square braces are the last thing that the scanner parses, and the file will start with another knot sequence afterwards.

4.2 Standard Input Format for Braids

In this section, we provide what the standard input format for braids are and what each item means. Files that we will scan and parse for braids are CMK files [3]. These are files of the standard input format that contains all of the existing braids, which look like:

braid(tr(7,3),[sigma(1, 2), sigma(2, 1), sigma(1, -1), sigma(2, 4)]).

Figure 4.2.1. input from braids_tr_7.cmk

The figure above is a sample line from braids_tr_7.cmk and is the braid sequence for the third variant knot with 7 crossings. Every item in this line has significant meaning. The braid_id in this line is denoted as tr(7,3). Tait-Rolfsen, which is denoted as tr, is the type of quandle that the braid uses. The number 7 is the total number of crossings that are in the knot, and 2 is the variant of the knot with 7 crossings. The sigma notation is used here to denote a crossing or multiple consecutive crossings. In the sigma notation, there are two numbers. The first number denotes a crossing between the ith and the $i+1$th position of the braid [3]. In this case, there are total 6 crossings because $1+2+1+2=6$, which is the sum of all the first numbers of the sigma notation. The second number of

the parenthesis tells us two things: whether if there are multiple consecutive crossings in the same position and if the crossing is inverse or not. For example, if we look at the first sigma notation, it is $sigma(1, 2)$. This means that there are two consecutive, non-inverse crossings between the first strand and the second strand.

4.3 Standard Input Format for Quandles

In this section, we provide what the standard input format for quandles are and what each item means. Files that we will scan and parse for quandles are CMK files [3]. These are files of the standard input format that contains all of the existing quandles, which look like:

```
quandles.
quandle(tait,
interpretation( 3, [number = 5,seconds = 0], [
    function(*(_,_), [
        0,2,1,
        2,1,0,
        1,0,2]),
    function(/(_,_), [
        0,2,1,
        2,1,1,
        1,0,2])])).
end_of_list.

knots.
torus(2,3).
torus(2,5).
end_of_list.
```

The figure above is the standard input format for the quandle for the Trefoil (3_1) knot. The first information we can get from this input file above is the quandle_id. The quandle_id is denoted after the word quandle, so the quandle_id in this case is tait. For the algorithm, we need to find out what the dimensions of the given quandle file is, and that is given to us after the token interpretation. In this case, it is 3. This makes sense because the Trefoil (3_1) knot only has 3 crossings. The last item we need from this file is the quandle itself. The star(*) table determines the div(/) table, so we only need to represent one of them.

5
Relational Database Management System

In this section, we describe what Relational Database Management System is, and how it is used in our case. Relational Database Management System, or RDBMS, is a common type of database that stores data in tables, so it can be used in relation to other stored datasets. Each table has a unique primary key, which is used to navigate and search items within a table. Other tables can use primary keys of other tables, otherwise known as foreign keys. More detail on the relational model can be found in [6]

5.1 Independent Entity Sets

In the Entity Relationship Diagram, or ERD, there are two connected independent entity sets: Braid Presentation and Knot Presentation. Each entity has a primary key: the Braid Presentation entity has braid_id, and Knot Presentation entity has knot_id. For each presentation table, it has dependent entity sets. There are arc_colours, crossings, and colouring entity sets. When we translate over to MySQL, we know the presentation tables are independent, and arc_colours, crossings, and colouring entity sets are dependent, because the dependent tables have foreign keys from where their dependency comes from.

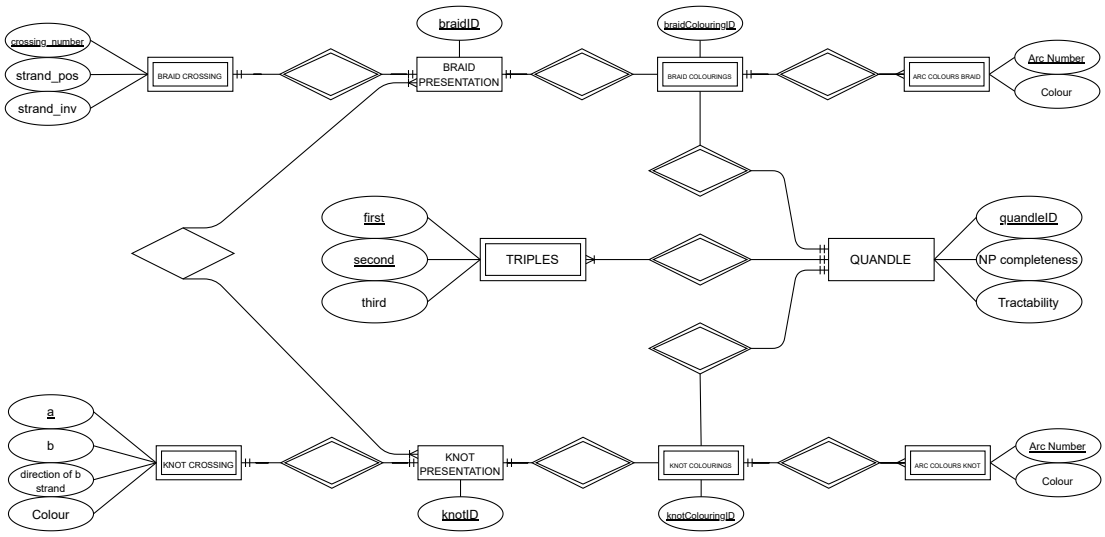

Figure 5.1.1. Entity Relationship Diagram

The figure above is the entity relationship diagram of the database, and this diagram provides foundational structure for the database, which carry numerous relationships. For example, the quandle entity is one of three entities that is independent, meaning, it can exist on its own. The triples entity, on the other hand, is a weak entity. Weak entities are denoted by two rectangles and their existence depends on other non-weak entities, and the triples entity depends on the quandle entity in this case because it is connected to the quandle entity and is not weak.

5.2 Crossings

There are two types of crossing entity sets in the ERD, each that depend on either presentation entity set. In the crossing entity that is dependent on braid presentation

entity, the attributes are crossing_number, strand_pos, and strand_inv. Since this entity depends on the braid presentation entity, the attribute of braid presentation can be used as well. As a result, the primary keys for this entity set are barid_id and crossing_number, and the braid_id is a foreign key. On the MySQL perspective, this translates very directly. Users will be able to tell braid_crossing table has dependency on braid_presentation table because braid_crossing table has a foreign key braid_id which references the braid_presentation table.

```
CREATE TABLE braid_crossing (
        strand_pos int not null,
        strand_inv int not null,
        crossing_number int not null,
        braid_id varchar(255) not null,
        PRIMARY KEY (crossing_number, braid_id),
        FOREIGN KEY (braid_id) REFERENCES braid_presentation(braid_id)
);
```

For the knot presentation entity, on the ERD, its relationship is same as that between braid presentation and braid crossing. Knot presentation entity only has one attribute, which is its primary key called knot_id. The attributes that knot crossing entity has are a_strand, b_strand, and b_direction, and the knot_id attribute can be passed down to knot_crossing due to its dependency. On the MySQL perspective, when users look at knot_crossing table and knot_presentation table, they will know that knot_crossing has dependency on knot_presentation, because knot_crossing table has two primary keys, knot_id and a_strand, where knot_id references the knot_presentation table.

```
CREATE TABLE knot_crossing ( -- originally crossing
        a_strand int not null, -- presented by the colour of the strand
        b_strand int not null,
```

```
        b_direction int not null,
        knot_id varchar(255) not null,
        PRIMARY KEY (a_strand, knot_id),
        FOREIGN KEY (knot_id) REFERENCES knot_presentation(knot_id)
);
```

5.3 Quandles and Triples

The quandle entity set is essential in this database because it acts like a bridge between knots and braids. the quandle entity set has three attributes: quandle_id, np_completeness, and tractability. The quandle_id is the primary key and will act as the most important attribute, because this attribute tells us what the shape of the knot is.

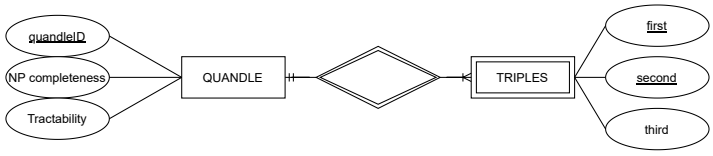

Figure 5.3.1. Quandle Entity Set

This is what the quandle table looks like in MySQL perspective:

```
CREATE TABLE quandle (
        quandle_id varchar(255) not null,
        np_completeness boolean,
        tractability boolean,
        PRIMARY KEY (quandle_id)
);
```

Notice that there is no indication of not null for the attributes np_completeness and tractability. That is because this is unknown for certain knots. This, however, is not true for quandle_id because quandle_id is the primary key of the quandle entity, therefore, it cannot be null.

Quandles are 2 dimensional tables that contain information about the knots. Its primary purpose is to show the users how the knot is formed. Quandles have two operations star(*) and div(/), and make two tables. However, the star(*) table determines the div table, so we only need to represent 1. The rows and columns of quandle represent the arc number. Here is an example quandle:

*	0	1	2
0	0	2	1
1	2	1	0
2	1	0	2

Table 5.3.1. quandle of trefoil (3_1) knot

The triples entity set is a weak entity set that branches out from the quandle entity set. We determined the triples entity set to be weak because triples are determined by quandles and cannot exist without quandles. The triples entity set contains three attributes first, second, and third. Notice that, first and second attributes are both primary keys. This is because each attribute in the triples entity set represent an arc colour, and one arc colour does not create a crossing. There needs to be two arcs in order for there to exist a crossing. The attribute first comes from the rows, the attribute second comes from the columns, and the attribute third comes as a result of first and second, these numbers determine where each crossing occurs in a knot. This is what the triples entity set looks like in the MySQL perspective:

```
CREATE TABLE triples (
        first int not null,
```

```
        second int not null,
        third int not null,
        quandle_id varchar(255) not null,
        PRIMARY KEY (first, second, quandle_id),
        FOREIGN KEY (quandle_id) REFERENCES quandle(quandle_id)
);
```

Notice that, the quandle_id is part of the triples entity set as a result of it being a weak entity set of the quandle entity set. We can verify this by looking at the last line of code, where quandle_id is identified as a foreign key that references the quandle table. The quandle previously mentioned will output 6 triples:

```
Tait, (0,1,2)
Tait, (0,2,1)
Tait, (1,0,2)
Tait, (1,2,0)
Tait, (2,0,1)
Tait, (2,1,0)
```

Tait denotes its quandle_id. Notice that, for an n x n quandle, there are $n^2 - n$ triples. There are total $n^2 - n$ triples in a because n x n quandle because we do not include the diagonal values, (0,0,0), (1,1,1), and (2,2,2). These are not triples because it just denotes idempotence, which is Reidemeister move Type I [5]. In this example, in a 3 x 3 quandle, there are $3^2 - 3 = 6$ triples.

5.4 Braid Colourings

Braid colourings have multiple dependencies within the ERD, and they are ultimately all related to each other, and these entities are arc_colours_braid, braid_crossing, and braid_presentation [6]. The arc_colours_braid entity has two attributes, arc number and

colour. Arc number is the primary key because each braid has a unique arc number, and the colour attribute defines what colour the arc number is. This entity set will be under braid_colourings entity set. Notice that, the relationship between braid_colourings and arc_colours_braid is a one-to-many relationship, and that is because there will be many arcs within a braid_presentation, and the braid_presentation entity, and there will be more than one arc within a braid_presentation.

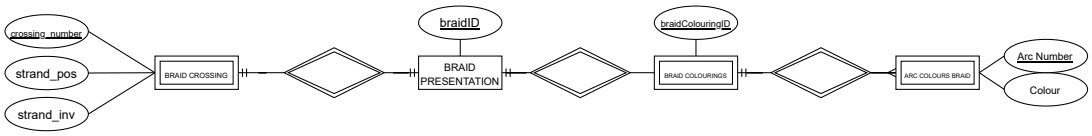

Figure 5.4.1. figure
Braid Entity Set

Here is the MySQL perspective of the all the braid entities:

```
CREATE TABLE braid_presentation (
        braid_id varchar(255) not null,
        PRIMARY KEY (braid_id)
);

CREATE TABLE braid_colouring (
        quandle_id varchar(255) not null,
        braid_colouring_id int not null,
        braid_id varchar(255) not null,
        PRIMARY KEY (braid_colouring_id, quandle_id, braid_id),
        FOREIGN KEY (quandle_id) REFERENCES quandle(quandle_id),
```

```
        FOREIGN KEY (braid_id) REFERENCES braid_presentation(braid_id)
);

CREATE TABLE arc_colours_braid (
        arc_number int not null,
        colour int not null, -- from the quandle
        quandle_id varchar(255) not null,
        braid_id varchar(255) not null,
        PRIMARY KEY (arc_number),
        FOREIGN KEY (quandle_id) REFERENCES quandle(quandle_id),
        FOREIGN KEY (braid_id) REFERENCES braid_presentation(braid_id)
);

CREATE TABLE braid_crossing (
        strand_pos int not null, -- subscript of sigma
        strand_inv int not null, -- superscript of sigma
        crossing_number int not null,
        braid_id varchar(255) not null,
        PRIMARY KEY (crossing_number, braid_id),
        FOREIGN KEY (braid_id) REFERENCES braid_presentation(braid_id)
);
```

The braid_presentation table acts as the core of all braid entities since it has the braid_id as its primary key, and the braid_id will be a foreign key in every other braid entity. The braid_crossing table has a one-to-one relationship [6]. The primary key here is the crossing_number and the braid_id, which is obtained from the braid_presentation table. The crossing_number attribute is the primary key in this table because each braid has multiple crossings, and each crossing needs to be kept track of. The braid_colouring table has a

one-to-one relationship with the braid_presentation table. The braid_colouring table acts as a bridge between the braid_presentation table and the arc_colours_braid table because different colours of braids will have to be identified differently. There will be many arcs within a braid, which makes the arc_colours_braid have a one-to-many relationship with the braid_colouring table, and these will be shown in the entity relationship diagram as arc number as the primary key with their colours accordingly.

5.5 Knot Colourings

Knot colourings, like braid colourings, have multiple dependencies within the ERD [6]. The entities that are interrelated are arc_colour_knot, knot_crossing, and knot_presentation. The arc_colours_knot entity has two attributes, which are arc number and colour. Arc number is the primary key of this entity because each knot has a unique arc number, and the colour attribute decides the colour of the arc number. This entity set will be dependent on knot_colourings entity set. Notice that, the relationship between knot_colouring and arc_colours_knot is a one-to-many relationship, and that is because there will always be more arcs within a knot_presentation, and the knot_presentation entity, and there will be more than one arc within a knot_presentation [6].

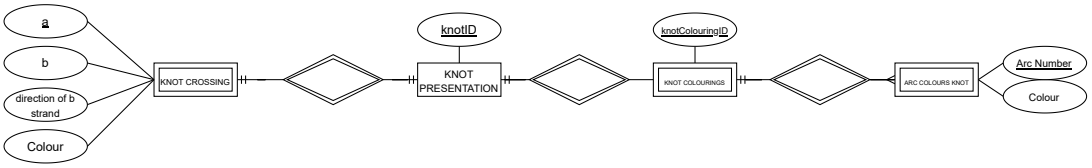

Figure 5.5.1. figure
Knot Entity Set

Here is the MySQL perspective of the all the knot entities:

```
CREATE TABLE knot_presentation (
        knot_id varchar(255) not null,
        PRIMARY KEY (knot_id)
);

CREATE TABLE knot_colourings (
        quandle_id varchar(255) not null,
        knot_id varchar(255) not null,
        knot_colouring_id int not null,
        PRIMARY KEY (knot_colouring_id, quandle_id, knot_id),
        FOREIGN KEY (quandle_id) REFERENCES quandle(quandle_id),
        FOREIGN KEY (knot_id) REFERENCES knot_presentation(knot_id)
);

CREATE TABLE arc_colours_knot (
        arc_number int not null,
        colour int not null, -- from the quandle
        quandle_id varchar(255) not null,
        knot_id varchar(255) not null,
        knot_colouring_id int not null,
        PRIMARY KEY (arc_number, knot_colouring_id, quandle_id, knot_id),
        FOREIGN KEY (knot_colouring_id) REFERENCES knot_colourings(knot_colouring_id),
        FOREIGN KEY (quandle_id) REFERENCES quandle(quandle_id),
        FOREIGN KEY (knot_id) REFERENCES knot_presentation(knot_id)
);

CREATE TABLE knot_crossing (
```

```
        a_strand int not null, -- presented by the colour of the strand
        b_strand int not null,
        b_direction int not null,
        knot_id varchar(255) not null,
        PRIMARY KEY (a_strand, knot_id),
        FOREIGN KEY (knot_id) REFERENCES knot_presentation(knot_id)
);
```

Just as the braid entities, the knot_presentation table acts as a basis for the other weak entities. We know this because the primary key of knot_presentation table is a foreign key of other tables. The braid_crossing table has a one-to-one relationship. The primary key here is the a_strand because all a_strands of the knot_crossing table will have different b_strands that crossovers, making it a unique property. The b_strand and the direction of the b_strand is important because it is the arc that crosses over the a_strand, and we need to know how it crosses over the a_strand. The knot_colouring table acts as a bridge between the knot_presentation table and the arc_colours_knot table because different colours of knots will have to be distinct. There will be many arcs within a knot, which makes the arc_colours_knot table have a one-to-many relationship with the knot_colouring table, and these will be noted in the entity relationship diagram as arc number as the primary key with their colours accordingly [6].

6
Code Translations

6.1 Braids

In this section, we describe an algorithm for storing braids in the database. First, we parse and get the necessary information for the braid_presentation table and braid_crossing table. In order to obtain information about braid_id, we start at the beginning of a new line and look for the word braid in a given token, because this denotes the start of a braid sequence. An example of a line of input from *braid_tr_7.cmk* is:

braid(tr(7,3),[sigma(1, 2), sigma(2, 1), sigma(1, -1), sigma(2, 4)]).

Figure 6.1.1. input from braids_tr_7.cmk

The braid_id in this line is tr(7,3). Tait-Rolfsen, which is denoted as tr, which is the type of quandle that the braid uses. The number 7 is the total number of crossings that are in the knot, and 2 is the variant of the knot with 7 crossings. The sigma notation is used here to denote a single crossing. In this case, there are total 6 crossings in the input above. For each sigma, the first number in the parenthesis denotes the position of the crossing between the ith and the $i+1$th index. The second number of the parenthesis tells us two things: whether if there are multiple consecutive crossings in the same positions and if it is inverse

or not. The next step is to individualise each crossing into single individual crossings. For example, the crossing $sigma(2,1)$ should be translated into $sigma(1,1), sigma(1,1)$, so that it is easier to parse and keep track of the total crossing number and the crossing number of the location of individual crossings. The code below is a simple algorithm to individualise each crossing.

```
numCross += Math.abs(inv);
if (inv > 1) {
        for (int i = 1; i <= inv; i++) {
                posandinv = posandinv.concat("sigma("+pos+",1) ");
        }
} else if (inv < -1) {
        for (int i = 1; i <= Math.abs(inv); i++) {
                posandinv = posandinv.concat("sigma("+pos+",-1) ");
        }
} else if (inv == 1 || inv == -1) {
        posandinv = posandinv.concat("sigma("+pos+","+inv+") ");
}
```

Here, numCross keeps track of the total number of crossing, so we can use it to set a limit on the amount of iterations to perform for a given braid sequence. We go through three conditionals, where the first two conditionals determine if the inv variable is greater one or less than -1. We iterate on the number quantity of inv there is saved in the variable, and appending it to a string. We do the same thing in the second conditional, but we take the absolute value of the inv variable, since inv will be less than -1, and appending it to a string. The third conditional is taking the inv value, which is 1 or -1, and appending it to a string. This string of newly created sigma notations and the total number of crossings is outputted onto a text file using the FileWriter class.

Using the outputted file, we scan and parse through this text file using a second scanner file for this outputted text file. This text file contains only contains information about the braid_id and their corresponding crossings that have been individualised crossings from the original input file. The following figure shows what the input line looks like after it has been scanned once:

braid(tr(7,3),[sigma(1,1), sigma(1,1), sigma(2,1), sigma(1,-1), sigma(2,1), sigma(2,1), sigma(2,1), sigma(2,1)]).

Figure 6.1.2. example input line after scanning braids_tr_7.cmk

At this point, there is no algorithm that needs to parse through this text file, so we proceed onto concatenating the information we need onto a string. We save the braid_id, pos, which is the first value of sigma, and inv, which is the second value of sigma, to a variable and concatenate this to a string, which is shown in the code below, and the resulting string will be written into a MySQL file as a series of DMLs using the FileWriter class.

```
int pos = Integer.parseInt(crossing.substring(0, crossing.indexOf(" ")).trim());
int inv = Integer.parseInt(crossing.substring(crossing.indexOf(" ")).trim());
output = output.concat("INSERT INTO braid_crossing(braid_id, crossing_number,
strand_pos, strand_inv) VALUES(\""+braid_id+"\","+numCross+","+pos+","+inv+");\n");
FileWriter writer = new FileWriter("braid_crossings_UPDATED.sql");
writer.write(output);
writer.close();
scan.close();
```

This will lead to one record in the braid_presentation table, which is highlighted in boldface:

braid_id
tr(7,1)
tr(7,2)
tr(7,3)
tr(7,4)
tr(7,5)
tr(7,6)
tr(7,7)

Table 6.1.1. braid_presentation table showing knots with 7 crossings

It will also create eight records in the braid_crossing table:

braid_id	crossing_number	strand_pos	strand_inv
tr(7,3)	1	1	1
tr(7,3)	2	1	1
tr(7,3)	3	2	1
tr(7,3)	4	1	-1
tr(7,3)	5	2	1
tr(7,3)	6	2	1
tr(7,3)	7	2	1
tr(7,3)	8	2	1

Table 6.1.2. braid_crossing table showing crossings for (7_3) knot

Each record in the braid_crossing table corresponds to one of the crossing terms in the input braid. Notice that, for a given crossing, the braid_id, strand_pos and strand_inv are simply taken directly from the braid expression. The crossing_number is the index of the crossings in the list.

6.2 Knots

Similar to how we parsed and got information for the braid_presentation and braid_crossing tables, we parse and get information for the knot_presentation table and knot_crossing table. Let us consider an example knot and describe its eventual representation from all-knots.pres file in the database:

presentation(3, 1, [-3, -1, -2]).

Figure 6.2.1. input from allknots.pres file for the trefoil 3_1

This is a line from the input file that represents the Trefoil (3_1) knot. The line starts off with the token presentation, which tells us that we are starting a new knot sequence. The number after, which is three in this case, represents the number of crossings in a knot, and the following number represents the invariant of the knot. The numbers in the square bracket tells us two things: b_strand that crosses over the a_strand, which is in numerical order, and the direction of the b_strand, which is also whether if the crossing is inverse or not.

```
if (num > 0) {
        if (inv == 1) {
                inv *= 1;
        } else {
                inv *= -1;
        }
} else {
        if (inv == 1) {
                inv *= -1;
        } else {
                inv *= 1;
        }
}
```

The code above parse through numbers in the input file, converting each token from a string value to an integer value and allowing the algorithm to determine whether the value of b_strand, which denotes the direction of the b_strand, is positive or negative. We only change the direction of the b_strand if the preceding direction of the b_strand is in the

opposite direction. Once all tokens of the line have been parsed, we tell the algorithm to reset and start parsing the next line of input.

Once we have the information about the knot_id, a_strand, b_strand, the direction of b_strand, we proceed to concatenating this to an empty string with MySQL data manipulation language (DML) [6], and we use the FileWriter class in Java to output all the DMLs onto a MySQL file for knot_presentation table and knot_crossing table, respectively:

```
output = output.concat("INSERT INTO knot_presentation(knot_id) VALUES(\""+knot_id
+ "\");\n");
output = output.concat("INSERT INTO knot_crossing(knot_id, a_strand, b_strand,
b_direction) VALUES(\"" + knot_id + "\","+numCross+","+Math.abs(num)+","+inv+");
\n");
FileWriter writer = new FileWriter("knot_crossings.sql");
writer.write(output);
writer.close();
scan.close();
```

This will lead to one record in the knot_presentation table, which is highlighted in boldface:

knot_id
presentation(0,1)
presentation(3,1)
presentation(4,1)
presentation(5,1)
...
presentation(10,162)
presentation(10,163)
presentation(10,164)
presentation(10,165)

Table 6.2.1. knot_presentation table showing all knots from the Tait-Rolfsen Knot Table

It will also produce three records in the knot_crossing table:

knot_id	a_strand	b_strand	b_direction
presentation(3,1)	1	3	-1
presentation(3,1)	2	1	-1
presentation(3,1)	3	2	-1

Table 6.2.2. knot_crossing table for trefoil(3_1)

Each record in the knot_crossing table corresponds to one of the list in the input. For a given crossing, the knot_id, b_strand, and b_direction are simply taken directly from the knot expression. In addition, notice that, the a_strand is the index of the b_strand in the presentation.

6.3 Quandles and Triples

In order to obtain everything we need to create proper DMLs [6] for quandles, we must have a few things. We need to first have the quandle_id. Just like how we scanned for the knot_id from *allknots.pres* and braid_id from *braids_tr_7.cmk*files, we need to scan and parse the file that contains information about an arbitrary quandle. In this example, we will use the knot quandle from *tait.cmk* file. In order to properly scan and parse, we need to know what how the standard input format for the quandle file is:

```
quandles.
quandle(tait,
interpretation( 3, [number = 5,seconds = 0], [
    function(*(_,_), [
        0,2,1,
        2,1,0,
        1,0,2]),
    function(/(_,_), [
        0,2,1,
```

```
        2,1,1,
        1,0,2])])).
end_of_list.

knots.
torus(2,3).
torus(2,5).
end_of_list.
```

Above is the standard input format for the quandle. The first information we need to obtain is the quandle_id, which we can see by looking at the first opening parenthesis, where it says "tait" after the parenthesis. That will be the quandle_id of this quandle. We then concatenate this information to a variable, which is used in creating SQL DML [6] for the quandle table:

```
sqldml = sqldml.concat("INSERT INTO quandle(quandle_id) VALUES(\""+quandle_id+"\");
\n");
```

The next essential information is the dimension of the quandle. We can obtain the information about the dimension of the quandle with the number, which is 3 in this case, after where it says interpretation. We look for a token that contains "interpretation," scan the proceeding token, and initialise the size of the quandle. The most important part about this algorithm is contained here, where we scan the quandle itself. Here, we only need to scan the first quandle, since both quandles output the same results. Since this information is contained in n different lines for n x n quandle, we have to put this is in one dimensional array and move it into a two dimensional array.

```
int[][] quandle = new int[dimension][dimension];
int[] tmpInt = new int[dimension * dimension];
int a = 0;
```

```
for (int r = 0; r < quandle.length; r++){
        for (int c = 0; c < quandle[r].length; c++){
                quandle[r][c] = tmpInt[a++];
                if (r == c) {
                        continue;
                } else {
                        sqldml = sqldml.concat("INSERT INTO
                        triples(quandle_id, first, second, third)
                        VALUES(\""+quandle_id+"\","+r+","+c+","+quandle[r][c]+");
                        \n");
                }
        }
}
```

The algorithm above allocates each different by iterating through the dimensions of the quandle row by row and column by column. Notice that, the dimension of tmpInt is dimesion $*$ dimension. This is because tmpInt is a one dimensional array that needs to be able to contain the output value of the crossing arithmetic of the row and column indices. As a result, we iterate through the quandle and allocate the values of tmpInt, which contain the output of the crossing arithmetic, in the correct order. Notice that there is a conditional in the inner for loop. The purpose of the quandle scanner is to obtain triples. From the code above, we would have total 6 triples:

```
Tait, (0,1,2)
Tait, (0,2,1)
Tait, (1,0,2)
Tait, (1,2,0)
Tait, (2,0,1)
Tait, (2,1,0)
```

These triples will form a part of the triples table in MySQL:

knot_id	first	second	third
0	0	2	1
1	2	1	0
2	1	0	2

Table 6.3.1. Tait-Rolfsen quandle for the trefoil(3_1) knot

7
Conclusion

In conclusion, we were able to successfully create a logical model for a knot-colouring database that can be transformed into a relational database. We were also successful in scanning and parsing the input files for the database and using the parsed information to create queries to populate the database. Note that, the standard input format for the input files will stay consistent, so the scanning and parsing program will work if the input format is not changed.

7.1 Future Works

For future research and work, it would be valuable to create an online system to store the database. In addition, the website could contain some interactive elements into it with the users. For example, one is where users are able to see all features about a specific knot, where the website displays the braid presentation of the knot, other knot variants of the same number of crossings, quandles, and the colourings. Another possible future work is coming up with a system to draw the braids on the fly with the given syntax. This will give drawings of braids in two dimensional projections and allow people to easily visualise them.

Bibliography

[1] D. Bar-Natan and S. Morrison, *The Knot Atlas*.

[2] W. McCune, *Mace4 reference manual and guide*, Technical Report ANL/MCS-TM-264, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 2003.

[3] R. W. McGrail and T. T. Nguyen, *Knot coloring as verification*, Synasc 2020, 2020, pp. 24–31.

[4] R. W. McGrail, T. T. Nguyen, T. T. Trang Tran, and A. J. Tripathi, *A terminating and confluent term rewriting system for the pure equational theory of quandles*, Synasc 2018, 2020, pp. 157–163.

[5] J. H. Przytycki, *3-coloring and other elementary invariants of knots*, Knot theory, 1998, pp. 275–295.

[6] J. D. Ullman and J. Widom, *A first course in database systems*, Vol. 3, 2008.