Solving a Combinatorics Challenge by Exploiting Computational Techniques Available on Relational Databases

Wei Hu
Software Engineering, Fairfield University
Fairfield, Connecticut USA
e-mail: wei.hu@student.fairfield.edu

Mirco Speretta
Gateway Community College
New Haven, Connecticut USA
e-mail: msperetta@gwcc.commnet.edu

Abstract—**Experimental studies are based on data that, sometimes, needs to be manually created. Moreover, the data is handled in relational databases to exploit their capabilities of manipulating (i.e., sorting, combining, and inserting) data. In this study, we show how this approach was successful in solving a combinatorics challenge to create a data set used in a separate research study that involves all the possible card combinations of the SET game®. The data required for the study was very extensive. The exact number was unknown, as this is an open combinatorics question, but the estimate was in the order of hundreds of millions. We solved this challenge by using a relational database (i.e., MySQL) as a computational tool to generate the data set. Advanced SQL scripts, based on cross joins, were applied to generate all the data. Table partitioning was also applied to improve the database performance of tables whose number of records exceeded the size capability of the database table. The data set created from this project was then used to support a Web based user interface that collects data to be used in a separate research study based on the SET® game.**

Keywords-MySQL; partitioning; computation; cross join.

I. Introduction

SET game® [1] is a popular card game created by Marsha Jean Falco in 1974. She is also the founder of Set Enterprises, Inc., the company that published the game in 1981. In this game, 12 cards (i.e., a hand), randomly selected from a deck of 81 cards, are placed in front of the players. The winner of a hand is the first player that identifies a group of three cards that makes a *SET*. There are four types of *SETs* that a player can identify.

Two professors from the department of mathematics, at Fairfield University, were responsible for a Math club in a middle school of the town. They incorporated the SET game into the sessions with the students. Noticing the selection of specific SETs by the students, the professors wanted to investigate further this behavior with a research study whose main goal was to explore whether the personal information of the player, such as gender, age, or academic interests, can influence the types of *SET* identified. The study is based on a statistical analysis of data collected from anonymous users that are playing the game using a Web based interface. To avoid any statistical bias in the study, each hand must include one (and only one) instance of each of the four types of *SET*. To give a rough estimate of the amount of data to be processed, first we needed to look at the total of possible combinations: given 81 cards (i.e., the deck) there are 7.07 $\times 10^{13}$ possible ways to choose a hand (i.e., 12 cards). Out of this number of combinations, we had to identify and remove all the hands that did not satisfy the requirement of the statistical design. This requirement presented the following two main challenges. The first challenge is the combinatorics challenge, which can be described as follows: it is not known how to count the total number of combinations of hands that comply with the requirements mentioned above. This is still an open question in the mathematical community. The second challenge is the technical challenge to guarantee efficiency, which can be described as follows: the number of combinations is too high and it would take too much time to select the cards that form a hand in real time; lots of time would be wasted generating combinations that do not comply with the requirement. Because of the two challenges explained above, the users of the Web interface would not be able to play the game properly.

The solution to both challenges was to pre-generate four types of *SET* to compose all the possible hand combinations and store them into a database table. This approach would allow to show a randomly picked hand of cards in real time.

In this paper, we describe the process of generating and storing the data set using the MySQL® [2] relational database. More specifically, in Section II, we list the software used in the study. In Section III, we provide the context to this study by describing specific features of the SET game. Section IV outlines the details of the methodology, along with the information about the final data generated. We conclude our work in Section V.

II. Background

MySQL [2] is a popular database that supports SQL along with transactions. The idea to use a relational database to manipulate data in our study comes from various research ideas, especially in the context of testing data [7].

We implemented our database on a MySQL 5.7.22 server, to store pre-generated hand data set. The server we used was equipped with 16 CPU (2.4 GHz) and 64 GB of RAM, running Linux Ubuntu (version 18.04.4 LTS). In our study, the largest amount of computational effort is needed to fulfill the following tasks: 1) joining tables to get the maximum number of possible card combinations; 2) validating that no more than one occurrence of a card appears in one hand; 3) validating that each hand card combination included only one instance of the four types of *SET*. Cross joins, comparisons between data tuples and row by row computations are all very time and resource consuming due to the big size of data. In this study, we show that all these tasks can be carried out using a relational database in a simple and efficient way.

III. THE SET GAME

The SET game is a popular card game that has been widely disseminated by online media such as the New York Times. It has been used in mathematics learning by several educational institutions at different school levels [3]-[6].

A. *Cards of the SET game*

The game SET has a rich mathematical structure based on combinatorics principles. An example of cards is shown in Figure 1.

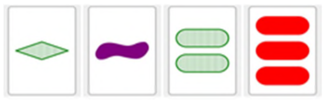


Figure 1. Typical cards of the SET game.

Cards have four attributes: number, shading, color, and shape. Each attribute has three features. The complete list is given in Table I.

TABLE I. SET CARD ATTRIBUTES AND THEIR VARIATIONS.

Attribute	Feature
Number	{One, Two, Three}
Shading	{Solid, Striped, Open}
Color	{Red, Green, Purple}
Shape	{Oval, Squiggle, Diamond}

The deck of the SET game has eighty-one cards, one for each possible combination of attributes.

B. *Rules of the SET game*

Three cards are called a *SET* if, with respect to each of the four attributes, the cards are either all the same or all different. The goal of the game is to find collections of three cards satisfying this rule. For example, the three cards in Figure 2 compose a *SET* because all cards have different shapes, different colors, and different shading, and each card has the same number of shapes (three).

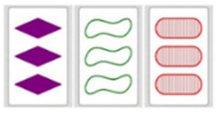


Figure 2. Typical *SET*.

C. *Four types of SETs*

Case 1: One attribute has different features; three attributes have the same features

a) different: shape; same: shade, color, number
b) different: shade; same: shape, color, number
c) different: color; same: shape, shade, number
d) different: number; same: shape, shade, color

Case 2: Two attributes have different features; two attributes have the same features

a) different: shape, shade; same: color, number
b) different: shape, color; same: shade, number
c) different: shape, number; same: shade, color
d) different: shade, color; same: shape, number
e) different: shade, number; same: shape, color
f) different: color, number; same: shape, shade

Case 3: Three attributes have different features; one attribute has the same feature

a) different: shape, shade, color; same: number
b) different: shape, shade, number; same: color
c) different: shape, color, number; same: shade
e) different: shade, color, number; same: shape

Case 4: All four attributes have different features

a) different: shape, shade, color, number

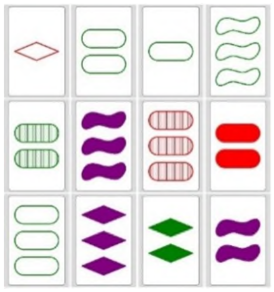


Figure 3. Typical hand of the SET game.

D. *Hands of the SET game*

To play the game, twelve cards, called a hand as shown in Figure 3, are dealt face up in front of players. Players search for *SETs*. After all *SETs* in the hand are found, the hand is refreshed and another twelve random cards are dealt out of the deck.

IV. METHODOLOGY

As the question on how to count the number of hand combinations containing exactly four *SET*s, one for each type, remains open, we could not know the exact number of records we were supposed to generate. The only solution to this problem was to work on an efficient algorithm to generate all the possible combinations.

Our goal is to find all possible hand card combinations that satisfy the requirements of the experimental study: every hand includes exactly four *SET*s, one for each type. In a typical play, 12 cards are dealt randomly out of the 81 cards deck. The total number of hand combinations to consider is about 7.07×10^{13}. This number of combinations is too big to be handled practically. For this reason, we had to apply some heuristic to reduce it to a number that was computationally feasible. If we start from the four types of *SETs* and we consider them as the basic components that form a hand, the biggest number of hand combinations to screen is no more than number of Type 1 *SET* × number of Type 2 *SET* × number of Type 3 *SET* × number of Type 4 *SET* = $432 \times 324 \times 108 \times 216 = 3{,}265{,}173{,}504 = 3.265 \times 10^9$. The number of each type of *SET* is shown in Table III. All sets for each type can be stored in a table and all four tables can be merged using cross joins. In this way, the four types of *SETs* are automatically combined to form all the possible hands. From this number, we need to remove the combinations that do not satisfy the requirements:

1. each card occurrence is unique in one specific hand.
2. no extra new *SET*s are formed other than the four built-in *SET*s.

We used the database server MySQL (version 5.7.22) to store and manipulate all the card combinations. The data flow from the generation of four types of *SETs* through the generation of the final hand combinations is illustrated in the following three steps.

Step 1: We implemented a Java program to generate the four basic tables storing the four types of *SET* (Figure 4).

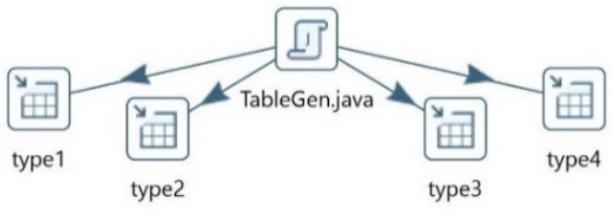


Figure 4. Java program generating the four basic tables.

Step 2: We implemented an SQL script to combine the four attributes of each card into a 4-digit number. This step also includes the generation of a new group of the four basic tables storing all the 4-digit numbers (Figure 5).

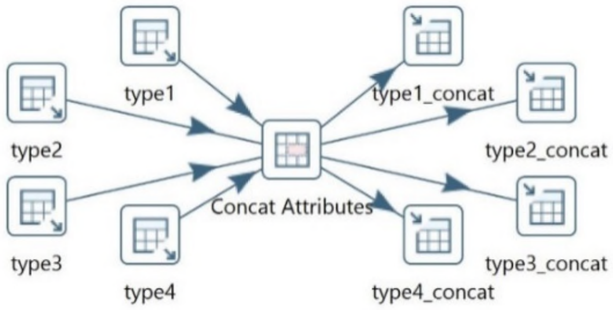


Figure 5. SQL script combining basic tables.

Step 3: We implemented an SQL script to cross join the four types of *SETs.* Then, we removed the records where any card occurred more than once, along with the records where the new *SET*s are formed (Figure 6).

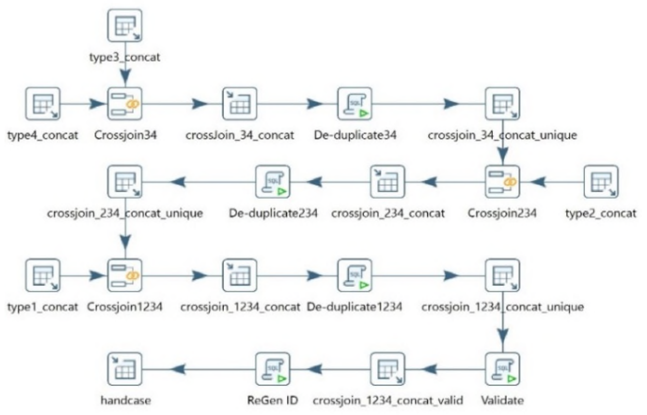


Figure 6. SQL script generating more SET hands and removing duplications.

In the remaining part of this section, we will provide more details about the work involved in carrying out the above steps.

A. Card Value Definition

We numbered all the cards using the following representation. A four-digit number is assigned to each card based on its specific attributes. From left to right, each digit refers to one attribute. Each digit can be either 0, 1, or 2. Each value represents one variable of an attribute (shown in Table II.) Using this representation, each card can assume a value in the range [0000] - [2222].

TABLE II. CARD DEFINITION.

Attribute / ***Variable***	**Number**	**Shading**	**Color**	**Shape**
	Position (from left to right)			
0	One	Open	Red	Diamond
1	Two	Striped	Green	Oval
2	Three	Solid	Purple	Squiggle

One example is shown in Figure 7.

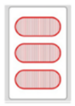


Figure 7. Visual representation of the card [2101].

This card value is [2101]: three, striped, red, and ovals. Another example is shown in Figure 8.

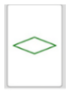


Figure 8. Visual representation of the card [0010].

This card value is [0010]: one, open, green, and diamond.

B. SET Type Definition

We define four types of *SET* as Type 1, Type 2, Type 3 and Type 4. The details of the definition are described below.

1) Type 1 *SET*: Only one attribute is the same, the other three attributes are different. The number of this type of *SET* is 432.

2) Type 2 *SET*: Two attributes are the same and the remaining two attributes are different. The number of this type of *SET* is 324.

3) Type 3 *SET*: Three attributes are the same, only one attribute is different. The number of this type of *SET* is 108.

4) Type 4 *SET*: All attributes are different. The number of this type of *SET* is 216.

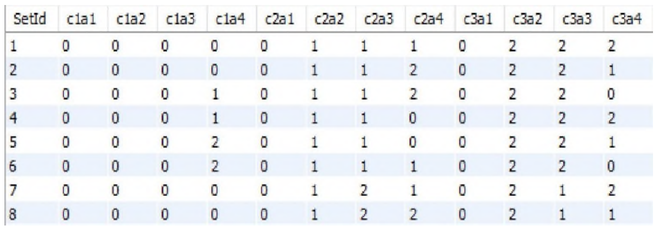
C. Four Basic Tables Creation

We created four basic tables, one table for each type of *SET*. Potential card combinations of hands are created using these four basic tables (Table III).

TABLE III. FOUR BASIC TABLES DEFINITION.

Table No.	Table Name	*SET* Type	Number of *SETs*
1	type1	Type 1	432
2	type2	Type 2	324
3	type3	Type 3	108
4	type4	Type 4	216

Every *SET* has three cards, namely Card1, Card2 and Card3. Each card has four attributes namely A1, A2, A3 and A4. We run a Java program to generate data for the four basic tables type1, type2, type3 and type4. We create one attribute of each card at one time. The structure of table is shown in Figure 9 (cNaM – card N attribute M, N = 1, 2, 3 M = 1, 2, 3, 4).



SetId	c1a1	c1a2	c1a3	c1a4	c2a1	c2a2	c2a3	c2a4	c3a1	c3a2	c3a3	c3a4
1	0	0	0	0	0	1	1	1	0	2	2	2
2	0	0	0	0	0	1	1	2	0	2	2	1
3	0	0	0	1	0	1	1	2	0	2	2	0
4	0	0	0	1	0	1	1	0	0	2	2	2
5	0	0	0	2	0	1	1	0	0	2	2	1
6	0	0	0	2	0	1	1	1	0	2	2	0
7	0	0	0	0	0	1	2	1	0	2	1	2
8	0	0	0	0	0	1	2	2	0	2	1	1

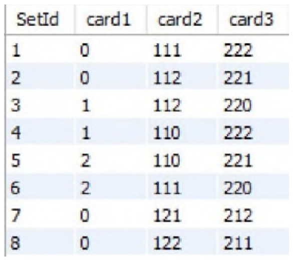
Figure 9. Basic SET table (type1-type4).

We then create and run an SQL script to validate the correctness of the data stored in these four tables.

In order to facilitate the calculation and improve the efficiency of the database, we combined the four attributes of each card into a one 4-digit number. The output of this process was to create the four new tables type1_concat, type2_concat, type3_concat, and tyep4_concat. They store the four types of *SETs* represented by the four-digit attribute value of cards. See a sample of these data in Figure 10.

Every *SET* is composed by three 4-digit numbers, each 4-digit number representing one card. For example: [0000,0111,0222] is a type 1 *SET*. The value of the first card is [0000] and refers to "one open red diamond".

Eight tables were created after validating and concatenating the data. For the remainder of the process, only the four tables with concatenated attributes were used.



SetId	card1	card2	card3
1	0	111	222
2	0	112	221
3	1	112	220
4	1	110	222
5	2	110	221
6	2	111	220
7	0	121	212
8	0	122	211

Figure 10. Basic SET table with concatenated attributes (type1_concat-type4_concat).

D. Cross join of the Four basic Tables and Deduplication

In order to work with the smallest possible amount of data, at any given time, we started the merging process using the two tables with the fewest number of records. They are represented by the tables type3_concat and type4_concat. Their joined table was then cross joined with type2_concat. As the last step, we cross joined this newly created table with the biggest table, type1_concat.

Due to the exponential increase of data size, after each cross join, we used SQL queries to validate records and filter out those records where the same cards were used more than once. In this study, this deduplicate validation is different from the typical deduplicate operation, which is responsible to remove redundant records from the table. We then performed another cross-join, the goal of which was to minimize the number of records of each cross join as much as possible. Table IV describes the number of records, the table size and the time spent to create each specific table. The tables whose names end by '_unique' refer to the tables created after the deduplication operation. In table IV we can notice the reduction in terms of both number of records and table size. The last table, 'crossjoin_1234_concat_valid', stored the valid hand combinations that were used in the research study.

TABLE IV. CROSS JOIN TABLES.

Table name	**Number of records**	**Table size (MB)**	**Time elapsed**
crossjoin_34_concat	23328	2	NS
crossjoin_34_concat_unique	20736	2	NS
crossjoin_234_concat	6718464	277	NS
crossjoin_234_concat_unique	5351040	216	NS
crossjoin_1234_concat	2311649280	118410	7h, 5m
crossjoin_1234_concat_valid	269635392	13121	27h, 40m

E. Hand Combination Validation

Once the four tables type1_concat, type2_concat, type3_concat, and type4_concat were cross joined into one table, the number of records grew fast. Although, upon the completion of each cross join, we removed the records where the instance of specific cards appeared more than once, still, a large amount of calculation had to be carried out to validate the *SET*s that were added. This was necessary because each group of three cards, one from each type of *SET*, could have made up a new *SET*. We used SQL queries again to validate which hand combinations contain exactly four *SET*s, one for each type excluding the new *SET*s added after the cross join operations. To improve the performance of the database and accelerate the speed of the calculation, we used the partitioning technique, built-in in MySQL, to organize each cross joined table into (~20) partitions.

TABLE V. POTENTIAL NEWLY FORMED *SET* COMBINATIONS.

Group	Potential ***SETs***		
Type 3	card1+card4+card7	card1+card4+card8	card1+card4+card9
+	card1+card5+card7	card1+card5+card8	card1+card5+card9
Type 4	card1+card6+card7	card1+card6+card8	card1+card6+card9
+	card2+card4+card7	card2+card4+card8	card2+card4+card9
Type 2	card2+card5+card7	card2+card5+card8	card2+card5+card9
	card2+card6+card7	card2+card6+card8	card2+card6+card9
	card3+card4+card7	card3+card4+card8	card3+card4+card9
	card3+card5+card7	card3+card5+card8	card3+card5+card9
	card3+card6+card7	card3+card6+card8	card3+card6+card9
Type 3	card1+card4+card10	card1+card4+card11	card1+card4+card12
+	card1+card5+card10	card1+card5+card11	card1+card5+card12
Type 4	card1+card6+card10	card1+card6+card11	card1+card6+card12
+	card2+card4+card10	card2+card4+card11	card2+card4+card12
Type 1	card2+card5+card10	card2+card5+card11	card2+card5+card12
	card2+card6+card10	card2+card6+card11	card2+card6+card12
	card3+card4+card10	card3+card4+card11	card3+card4+card12
	card3+card5+card10	card3+card5+card11	card3+card5+card12
	card3+card6+card10	card3+card6+card11	card3+card6+card12
Type 3	card1+card7+card10	card1+card7+card11	card1+card7+card12
+	card1+card8+card10	card1+card8+card11	card1+card8+card12
Type 2	card1+card9+card10	card1+card9+card11	card1+card9+card12
+	card2+card7+card10	card2+card7+card11	card2+card7+card12
Type 1	card2+card8+card10	card2+card8+card11	card2+card8+card12
	card2+card9+card10	card2+card9+card11	card2+card9+card12
	card3+card7+card10	card3+card7+card11	card3+card7+card12
	card3+card8+card10	card3+card8+card11	card3+card8+card12
	card3+card9+card10	card3+card9+card11	card3+card9+card12
Type 4	card4+card7+card10	card4+card7+card11	card4+card7+card12
+	card4+card8+card10	card4+card8+card11	card4+card8+card12
Type 2	card4+card9+card10	card4+card9+card11	card4+card9+card12
+	card5+card7+card10	card5+card7+card11	card5+card7+card12
Type 1	card5+card8+card10	card5+card8+card11	card5+card8+card12
	card5+card9+card10	card5+card9+card11	card5+card9+card12
	card6+card7+card10	card6+card7+card11	card6+card7+card12
	card6+card8+card10	card6+card8+card11	card6+card8+card12
	card6+card9+card10	card6+card9+card11	card6+card9+card12

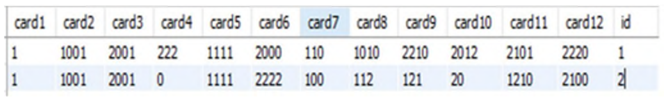
We represented a hand, including the four types of *SETs,* with the labels card1, card2, card3 through card12. In these 12 cards, there are 4 groups (i.e., 27 3-card combinations per group) that could make up new *SETs* (Table V). We need to check all the possible 3-card combinations (27 × 4 = 108) to filter out those hands that include extra *SETs* (i.e., more combinations than the four built-in *SETs* that are required.)

TABLE VI. ATTRIBUTE CHECK OF 3-CARD COMBINATION.

Color attribute of 3 cards	**All different**	**All the same**	**Neither (Non-SET)**
Attribute value of 3 cards	0+1+2	All 0, 1 or 2	0+0+1, 0+0+2, 1+1+2, 1+1+0, 2+2+0, 2+2+1
Sum of attribute value	3	0, 3 or 6	1, 2, 4 or 5
Remainder (Sum%3)	0	0	1 or 2

Each card has four attributes, for example [0000]. To validate a 3-card combination, we need to check each of their attributes. Only when three cards are either all-the-same or all-different with respect to their four attributes, they form a *SET*. If any attribute of 3 cards is neither all-the-same nor all-different, the 3-card combination is not a *SET*. Let us consider the color attribute of three cards to explain how to check each attribute. TABLE VI illustrates this process.

We summed up the value of the color attributes of three cards, then we divided the sum by three and we looked at the remainders. When the attributes are either all different or all the same, then the remainders are zero. Otherwise, the remainder is either one or two.



card1	card2	card3	card4	card5	card6	card7	card8	card9	card10	card11	card12	id
1	1001	2001	222	1111	2000	110	1010	2210	2012	2101	2220	1
1	1001	2001	0	1111	2222	100	112	121	20	1210	2100	2

Figure 11. Typical hand records before validation

Let us consider two hand records, shown in Figure 11 as an example to illustrate how to check four attributes of 3-card combinations. From the two hand records shown in Figure 11, we took a 3-card combination namely card1, card4 and card7 of each hand (shown in Table VII.). For each hand, we summed up the corresponding four attributes of three cards to get four sums, then divided the four sums by three. Only when the four remainders are all zeros, the 3-card combination forms a SET. Otherwise, they are not a SET. Table VII shows the details of the validation, where A1 refers to attribute 1 of the card, and so on.

TABLE VII. 3-CARD COMBINATION VALIDATION

Hand (id=1)	**A 1**	**A 2**	**A 3**	**A 4**	**Hand (id=2)**	**A 1**	**A 2**	**A 3**	**A 4**
Card 1	0	0	0	1	Card 1	0	0	0	1
Card 4	0	2	2	2	Card 4	0	0	0	0
Card 7	0	1	1	0	Card 7	0	1	0	0
Sum	0	3	3	3	**Sum**	0	1	0	1
Remainder (Sum%3)	0	0	0	0	**Remainder (Sum%3)**	0	1	0	1
	SET					***Non-SET***			

Next, we converted this algorithm into an SQL script that can be run on the database to validate every hand by checking each 3-card combination that could form a *SET*. As an example, let us consider the validation of the 3-card combination of card 1, card4 and card7. We used the condition sentence of *Not (((card1+card4+card7) div 1000) mod 3) + (((card1+card4+card7) div 100) mod 10 mod 3) +(((card1+card4+card7) div 10) mod 10 mod 3) + ((card1+card4+card7) mod 10 mod 3) = 0* to validate that the 3-card combination is not a *SET*; and added the condition in a WHERE clause of a SELECT statement to retrieve the records that do not include new *SETs*.

After validation, those hand combinations having extra *SETs* other than the four built-in ones were all filtered out. Finally, we achieved 269,635,392 hand combinations that were meeting our requirements and stored them into a table.

V. CONCLUSIONS

Experimental studies, which are based on the users' feedback, face various challenges. The main goal of this study was about answering the question whether the identification of *SET*s in the card game SET is to be linked to personal traits. In this paper, we tackled the specific problem of generating the data used to support a user facing Web application that was required in the process of collecting the experimental data.

The amount of data to be generated was very considerable and presented a challenge since it was not possible to count mathematically the number of card combinations (i.e., hands) to consider. By implementing our computational design into a relational database server, we were able to generate all the card combinations required in the Web based user interface.

Our process solved the following two main problems. The first one was about defining the appropriate representation of the cards in the SET game. This task required to consider minimal memory usage and quick validation of SET card combinations. The second challenge was about using a database server that could handle the required amount of data and could easily generate SET cards combinations by applying advanced SQL scripts.

Not only our methodology was successful in generating the required data, but also provided a computational answer to the mathematical challenge of providing the counts of hands combinations. We believe that this approach can be used in many other scenarios in which the creation of data generation is required. Experimental studies based on users' feedback should particularly benefit from this approach.

ACKNOWLEDGMENT

We would like to thank Dr. Janet Striuli and Dr. Laura McSweeney for sharing their research idea and providing support throughout the project. We would also like to express our gratitude to the faculty and students of Fairfield University and Gateway Community College for their commitment to our data collection. A special mention to the Physics department of Fairfield University to let us use their server where we were able to run our code. Thanks also to Set Enterprises, Inc. for allowing us to implement our study based on their SET game. Finally, we would like to thank Dr. Rankin and the Graduate Student Research Committee for providing financial support to our study.

REFERENCES

[1] PlayMonster. LLC, "SET - PlayMonster," [Online]. Available: https://www.playmonster.com/product/set/. [Accessed 07 2020].

[2] Oracle Corporation, "MySQL," [Online]. Available: https://www.mysql.com/. [Accessed 07 2020].

[3] B. L. Davis and D Maclagan, "The Card SET game," *The Mathematical Intelligencer,* vol. 25, no. 3, pp. 33-40, 2003.

[4] J. Vinci, "The maximun number of SETs for N cards and the total number of interal SETs for all partitions of the deck," June 2009. [Online]. Available: https://www.setgame.com/sites/default/files/teacherscorner/SETPROOF.pdf. [Accessed 07 2020].

[5] P. J. Fogle, "SET® AND MATRIX ALGEBRA," 15 03 2019. [Online]. Available: https://www.setgame.com/set-and-matrix-algebra. [Accessed 07 2020].

[6] N. Taatgen, M. van Oploo, J. Braaksma, and J. Niemantsverdriet, "How to construct a believable opponent using cognitive modeling in the game of Set," in *The fifth international conference on cognitive modeling,* pp. 201-206, Bamberg, 2003.

[7] C. De La Riva, M. J. Suárez-Cabal, J. Tuya, "Constraint-based test database generation for SQL queries," in *Proceedings of the 5th Workshop on Automation of Software Test (i.e., AST)* pp. 67–74, 2010.