Permutations, Combinations, Sequences and Random Numbers

Building Complex Numbers Tables

Scott Peters

https://advancedsqlpuzzles.com/

Last Updated: 07/05/2022

Welcome

I am back with a new set of puzzles. I hope you enjoy these puzzles as much as I have enjoyed creating them!

Solutions to these puzzles are located in the following GitHub repository:
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

All solutions are written in SQL Server's T-SQL.

To view my other puzzles and general database musings:
https://advancedsqlpuzzles.com/

As the title of this document suggests, I have collected all the puzzles related to permutations, combinations, and sequences into one document. Solving these puzzles will require a deeper knowledge of your SQL thinking, focusing on such constructs as using recursion and control structures to reach the desired output. Ultimately these puzzles resolve to creating number tables, which can be used to fill in gaps, create ranges and tallies, provide custom sorting, and allow you to create set based solutions over iterative solutions.

The solutions are provided in the GitHub repository and can be used as recipes for your future programing needs. I also present these problems in puzzle form in hopes someone discovers more formidable solutions to these puzzles. I have tried to include various ways the puzzles can be solved, but there may be other ways such as using the HEIRARCHYID data type or creating dynamic SQL based upon the number of cross joins needed (not sure I would recommend this solution, but it would be a solution). I also tried to provide the most parameterized solution possible, but puzzles like the Four-Vehicle puzzle are very much coded to the requirements.

One puzzle that you will notice is missing from this collection is a Sudoku puzzle. Solving a Sudoku via SQL is probably the most challenging of puzzles and considered by many to be the pinnacle of SQL puzzles to be solved, but I have decided to forego this as I want to deal with a bit of pragmatism. A puzzle book such as a PennyPress Dell Puzzle Book (or something similar) has an assortment of puzzles to solve that deal with permutations, combinations, and sequences. In the future, I hope to start including these puzzles as well into this document, but for now the puzzles I have provided here will have to suffice.

I welcome any corrections, new tricks, new techniques, dead links, misspellings, bugs, and especially any new puzzles that would be a great fit for this document. Please contact me through the contact page on my website.

The latest version of this document can be found below, or if you need to link to the GitHub repository: https://advancedsqlpuzzles.com/

On to the puzzles...

Puzzle #1
Factorials

Create a numbers table of the numbers 1 through 10 and their factorial.

Here is the expected output.

Number	Factorial
1	1
2	2
3	6
4	24
5	120
6	720
7	5,040
8	40,320
9	362,880
10	3,628,800

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

---

Puzzle #2
All Permutations

Create a numbers table of all permutations of *n* distinct numbers.

Here is the expected output for the set of {1, 2, 3}.

MaxNumber	Permutation
3	1,2,3
3	1,3,2
3	2,1,3
3	2,3,1
3	3,1,2
3	3,2,1

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

---

Puzzle #3
Growing Numbers

Create a numbers table that shows all numbers 1 through *n* and their order gradually increasing by the next number in the sequence.

Here is the expected output where *n* = 5.

Permutation
1
12
123
1234
12345

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

---

Puzzle #4
Non-Adjacent Numbers

Given an ordered set of numbers (for example {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}); create a result set of permutations where no two adjacent entries are adjacent numbers.

For example…

The arrangement {1, 3, 5, 7, 9, 2, 4, 6, 8, 10} **would** fit the criteria as no two entries are adjacent numbers.

The arrangement {**1**, **2**, 4, 6, 8, 10, 3, 5, 7, 9} would **not** fit the criteria as 1 and 2 are adjacent numbers.

The arrangement {1, 4, 2, 6, 7, 10, 3, 5, **8**, **9**} would **not** fit the criteria as 6 and 7 are adjacent numbers.

The arrangement {1, **3**, **2**, 6, 7, 10, 9, 5, 8, 4} would **not** fit the criteria as 3 and 2 are adjacent numbers.

Here is the expected output for the set of {1, 2, 3, 4}.

Permutation
2,4,1,3
3,1,4,2

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

---

Puzzle #5
Add the Numbers Up

Given an ordered set of numbers 1 through n (for example: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10), and a + or – sign at all possible groupings; create all possible permutations and the amount in which they add up to.

Here is the expected output for the set of {1, 2, 3}.

Permutation	Sum
123	123
1+2+3	6
1+2-3	0
1+23	24
1-2+3	2
1-2-3	-4
1-23	-22
12+3	15
12-3	9

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

---

Puzzle #6
Permutations of 0 and 1

Create a result set of all permutations for the combination of 0 and 1 with a length of *n* digits?

Here is the expected output for permutations with a length of 3 digits.

Permutation
000
001
010
011
100
101
110
111

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

Puzzle #7
Permutations 1 through 10

Display all permutations for the numbers 1 through *n*, displaying only the first *n* numbers.

Here is the expected output for all 24 permutations for the set {1,2,3,4}, displaying only the first three numbers.

Permutation
1,2,3
1,2,4
1,3,2
1,3,4
1,4,2
1,4,3
2,1,3
2,1,4
2,3,1
2,3,4
2,4,1
2,4,3
3,1,2
3,1,4
3,2,1
3,2,4
3,4,1
3,4,2
4,1,2
4,1,3
4,2,1
4,2,3
4,3,1
4,3,2

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

Puzzle #8
Four Vehicles Problem

https://advancedsqlpuzzles.com/

Here is an example problem involving combinations.

Given the following four vehicles:

- 1-seat motorcycle
- 2-seat sidecar
- 3-seat golf cart
- 4-seat car

There are 10 people total; 5 are children, 5 are adults. Only an adult can drive a vehicle.

Create a table of all possible 7,200 arrangements, assuming seating order does not matter.

We can determine there are 7,200 arrangements by using the following equation.

$$\text{Total Arrangements} = \frac{5!}{1!} * \frac{6!}{3!*2!*1!*0!} = 7{,}200$$

Solutions to the puzzles are located in the following GitHub repository.
[AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers]

Puzzle #9

Find The Spaces

Given the following table of SQL statements, provide a numbers table that displays a summary of the space character for each SQL statement.

Statement
SELECT EmpID FROM Emps;
SELECT * FROM Trans;

Here is the expected output.

RowNumber	Id	String	Starts	Position	Word	TotalSpaces
1	2	SELECT * FROM Trans	1	7	SELECT	3
2	2	SELECT * FROM Trans	8	9	*	3
3	2	SELECT * FROM Trans	10	14	FROM	3
4	2	SELECT * FROM Trans	15	0	Trans	3
1	1	SELECT EmpID FROM Emps;	1	7	SELECT	3
2	1	SELECT EmpID FROM Emps;	8	13	EmpID	3

3	1	SELECT EmpID FROM Emps;	14	18	FROM	3
4	1	SELECT EmpID FROM Emps;	19	0	Emps;	3

Solutions to the puzzles are located in the following GitHub repository.
[AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers]

Puzzle #10
Seating Chart

Given the following set of integers, write an SQL statement to determine the expected outputs.

```
CREATE TABLE #SeatingChart
(
SeatNumber INTEGER
);
GO

INSERT INTO #SeatingChart VALUES
(7),(13),(14),(15),(27),(28),(29),(30),
(31),(32),(33),(34),(35),(52),(53),(54);
GO
```

Here is the expected output.

Gap Start	Gap End
1	6
8	12
16	26
36	51

Total Missing Numbers
38

Type	Count
Even Numbers	8
Odd Numbers	9

Solutions to the puzzles are located in the following GitHub repository.
[AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers]

Puzzle #11
Count the Groupings

Write an SQL statement that counts the consecutive values in the Status field.

Step Number	Status
1	Passed
2	Passed
3	Passed
4	Passed
5	Failed
6	Failed
7	Failed
8	Failed
9	Failed
10	Passed
11	Passed
12	Passed

Here is the expected outcome.

Min Step Number	Max Step Number	Status	Consecutive Count
1	4	Passed	4
5	9	Failed	5
10	12	Passed	3

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

---

Puzzle #12
Double or Add 1

Create a numbers table where you start with the number 1, and then double the number if the result is less than 100, else add 1.

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

---

Puzzle #13

https://advancedsqlpuzzles.com/

Pascal's Triangle

Solve for any position in Pascal's Triangle.

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

---

Puzzle #14
Josephus Problem

Solve the Josephus Problem.

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

---

Puzzle #15
High-Low Card Game

Write a program that shuffles a standard deck of cards and plays a game of High-Low.

The game is played by receiving an initial card and then determining if the next card will be of higher or lower value based upon probability. Make a random decision of higher or lower where necessary.

Document 10,000 iterations through a deck of cards and display the number of correct guesses for each iteration.

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

---

Puzzle #16
Monty Hall Simulation

Run 10,000 simulations of the Monty Hall problem to prove it is true.

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

---

Puzzle #17
Dice Throw Game

Given 1 million trials, what is the average number of dice throws needed to reach 100 points given the following rules?

- Starting at 0, for each dice throw resulting in 1 through 5, add the dice amount to your score.
- If you roll a 6 (even on a re-roll), re-roll the dice and reduce your score by this amount. You cannot go lower than 0 points.

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

---

Puzzle #18
The Birthday Problem

Run 10,000 simulations of the Birthday problem to prove it is true.

Solutions to the puzzles are located in the following GitHub repository.
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

---

Overview

As the title of this document implies, the solutions to these puzzles require the use of permutations, combinations, sequences, and random numbers to solve. The following is a brief overview of some relevant talking points concerning these topics that I want to highlight.

Warning, you may consider some of the discussion below to be a spoiler alert on how to best solve these puzzles.

Solutions to these puzzles are located in the following GitHub repository:
AdvancedSQLPuzzles/Permutations, Combinations, Sequences and Random Numbers

All solutions are written in SQL Server's T-SQL.

To view my other puzzles and general database musings:
https://advancedsqlpuzzles.com/

Creating a Simple Numbers Table

At the heart of these puzzles a numbers table is ultimately involved in creating the solution. Numbers tables are much like calendar tables, and can be used to fill in gaps, create ranges and tallies, provide custom sorting, and allow you to create set based solutions over iterative solutions.

A numbers tables can be created using recursion, and you will find that many of these puzzles can be solved using recursion for certain (if not all) aspects of the puzzle.

Here is the code to create a numbers table using recursion.

```
DECLARE @vTotalNumbers INTEGER = 100;

WITH cte_Number (Number)
AS (
    SELECT 1 AS Number
    UNION ALL
    SELECT  Number + 1
    FROM   cte_Number
    WHERE  Number < @vTotalNumbers
    )
SELECT Number,
       (CASE Number WHEN 1 THEN 1 ELSE NULL END) AS Calculation
INTO   #Numbers
FROM   cte_Number
OPTION (MAXRECURSION 0)--A value of 0 means no limit to the recursion level
```

Here is another little trick to create a numbers table in SQL Server. This will only work in a SQL script, as the GO command is not a Transact-SQL statement.

```
SET NOCOUNT OFF;
DROP TABLE IF EXISTS #Numbers;
GO

CREATE TABLE #Numbers
(Number INT IDENTITY(1,1) PRIMARY KEY,
 InsertDate DATETIME NOT NULL);
GO

INSERT INTO #Numbers (InsertDate) VALUES (GETDATE())
GO 100
```

Per Microsoft documentation, GO is not a Transact-SQL statement; it is a command recognized by the sqlcmd and osql utilities and SQL Server Management Studio Code editor. SQL Server utilities interpret GO as a signal that they should send the current batch of Transact-SQL statements to an instance of SQL Server.

Permutations and Combinations

A permutation is a way in which a set or number of things can be ordered or arranged.

With permutation, order matters. With combinations, order does not matter.

We often use the work combination loosely without thinking order is important. A good example is the following:

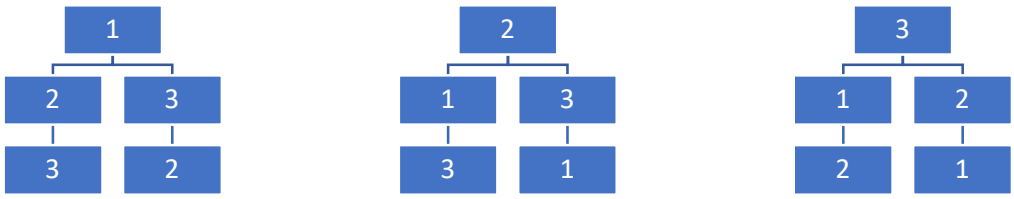
- "My cheeseburger has a combination of the toppings; lettuce, tomato, and onion". Here we really do not care about order, it is the same cheeseburger if it was "onion, lettuce, and tomato" or "tomato, lettuce, and onion". The same cheeseburger is described no matter the order of the ingredients.
- "The combination to my locker is 23-56-12". Here order is important, and a combination lock is more accurately described as a "permutation lock". A different arrangement would yield an inaccurate result for opening the lock.

Permutations are hierarchies

You can check out the Wikipedia article here on graph theory, or better yet, Joe Celko has a great book called "Trees and Hierarchies in SQL for Smarties" that is a great resource for understanding graph structures related to SQL. Mr. Celko does not go specifically go into permutations in this book, but this will give you the foundation for understanding such concepts as cyclic and acyclic data structures.

Because permutations are hierarchies, permutation problems can be solved using recursion, which are self-referencing common table expressions. In the solutions I have provided, I have tried (for the most part) to find the most elegant solutions using recursion.

To best understand that permutations are hierarchies, consider the following graph representations of the set {1,2,3} and the 6 different possibilities we can order these numbers.



The beginning or starting number of each permutation would be the root, followed by a branch for each possibility. From the above we get the following 6 permutations, which we know to be correct as we have 3 numbers and the factorial of 3 equals 6.

The permutations being {1,2,3}, {1,3,2}, {2,1,3}, {2,3,1}, {3,1,2}, {3,2,1}.

Permutations can get very large!

When calculating permutations, the number of permutations grows substantially with each additional element added. The number of permutations to arrange the numbers 1 through 9 can be calculated by 9! (9 factorial) and returns 362,880 permutations. Adding another element to this set (10!) causes the number of permutations to grow to 3,628,800.

When solving these puzzles, consider using a comma separated list to store the data (often referred to as zero normal form) and when needed you can pivot the data, which you have several options for. The solution to Puzzle #9 "Find the Spaces" can give you a recipe for pivoting the data using recursion or you can use the STRING_SPLIT function to perform this action. Please check out the documentation for the STRING_SPLIT function, as it does have several parameters. One particular note is the ENABLE_ORDINAL argument and ordinal output column are currently only supported in Azure SQL Database, Azure SQL Managed Instance, and Azure Synapse Analytics (serverless SQL pool only). It will not work in your Desktop SQL Server Express edition.

Incorporating multi-digit numbers

When solving the puzzles, your working data should include numbers of both single and multi-digit numbers, such as the set 1 through 10. This will become more clearer as you work through the puzzles, but certain solutions will require the use of the CHARINDEX function which searches for a substring in a string and returns the position. When using this function, you need to account that the number 1 is not the digit 1 in the number 10 or the number 11.

If you have problems solving for this scenario, consider using a mapping table where you assign single ASCII characters to your numbers, and then use the REPLACE function to update the characters to the desired number. An example of this is the numbers 1 through 10 can be replaced with the letters A through J, and then simply use the REPLACE function at the end of your solution to update this value.

Sequences

Microsoft's T-SQL can create a SEQUENCE object, which creates a list of ordered numbers. The sequence of numeric values is generated in an ascending or descending order at a defined interval and can be configured to restart when exhausted. It acts much like an IDENTITY column, but a SEQUENCE object is schema bound (i.e., You use the syntax CREATE SEQUENCE to define the object).

The benefit of the SEQUENCE object is that generated values can be used across multiple tables or columns, and you can recycle the values and restart from the minimum value.

The downside of using the SEQUENCE object is that it is 1) schema bound and 2) does not accept parameterization. The parameters (such as min and max values) must be hardcoded to a number, as you cannot set these via a variable. You can use dynamic SQL to create the SEQUENCE object to

circumvent this issue, but it may be best to create your own sequence table using a WHILE loop (while also avoiding creating a schema bound object as you can use a temporary table).

A real-world example of using a sequence object is where you need to display a quote on a website for each day where you must cycle through the quotes when exhausted. Here you could create the following table that joins a calendar table, a sequence table, and a quotes table together.

Calendar Date	Sequence ID	Quote
01/01/2023	1	Live long and prosper!
01/02/2023	2	I am a doctor, not a bricklayer.
01/03/2023	3	Beam me up, Scotty!
01/04/2023	1	Live long and prosper!
01/05/2023	2	I am a doctor, not a bricklayer.
01/06/2023	3	Beam me up, Scotty!
01/07/2023	1	Live long and prosper!
01/08/2023	2	I am a doctor, not a bricklayer.
01/09/2023	3	Beam me up, Scotty!

The SQL to generate the above table would be the following.

```
SELECT c.CalendarDate,
       s.SequenceNumber,
       q.Quote
FROM   CalendarTable c INNER JOIN
       Sequence s on c.RowNumber = b.RowNumber INNER JOIN
       Quotes q   on s.SequenceNumber = q.RowNumber;
```

Random Numbers

Microsoft provides the RAND function to generate a pseudo-random float value from 0 through 1, exclusive, and the NEWID function to create a unique value of type UNIQUEIDENTIFIER. Although not explicitly stated in the Microsoft documentation, the value returned from the NEWID function can be used as a random value.

The benefit of using NEWID over the RAND function is that NEWID will return a different but unique value for each record in the table, giving you the ability to randomize the order of a table. When used in an SQL statement, the RAND function will return the same value for each record in the table.

The benefit of the RAND function is that it provides a seed argument. If the seed argument is left blank, the database engine assigns a seed value at random and for a specified seed value, the result returned will always be the same.

If you need a random integer (for example you are simulating dice rolls), there are numerous ways in SQL to return an integer value (remember the RAND function returns a float value, not an integer). A quick internet search will return a multitude of statements that return an integer value, and some of them are not truly random!

I recommend the following syntax to generate random integers between 1 and n.

```
ABS(CHECKSUM(NEWID()) % n) + 1
```

Before you implement any type of random integer solution, ensure the statement returns a random number without any bias!

To determine if the random number generator truly is random, use the law of large numbers and generate enough numbers where the distribution should be even across all the numbers. For example, if you need to simulate dice rolls, run the random number generator 1,000 times, and ensure each value appears 1/6 of the time. If you want to measure the randomness using statistical measures, the Chi-Square test can be used to assess the goodness of fit between observed and expected values.

Also, I recommend saving your random number generator as a function and create a stored procedure to validate the randomness of the function (which can then be reused when a new user needs to ensure it is truly random).

Here is a quick script that creates a 1 million random numbers and checks the count of each number and the percentage of its occurrence.

```
SET NOCOUNT OFF;
DROP TABLE IF EXISTS #Numbers;
GO

CREATE TABLE #Numbers
(Number INT IDENTITY(1,1) PRIMARY KEY,
 InsertDate DATETIME NOT NULL);
GO

INSERT INTO #Numbers (InsertDate) VALUES (GETDATE())
GO 100000

;WITH cte_RandomNumber AS
(
SELECT  ABS(CHECKSUM(NEWID()) % 10) + 1 AS RandomNumber
FROM    #Numbers
)
SELECT  RandomNumber,
        COUNT(RandomNumber) AS RandomCount,
        COUNT(RandomNumber) / CAST((SELECT MAX(Number) FROM #Numbers) AS FLOAT) AS
Perc
FROM   cte_RandomNumber
GROUP BY RandomNumber
```

When solving such puzzles as the Dice Roll Game, many developers will think of creating a fully iterative based solutions rather than a set-based solution. The developer will write the code to roll the dice, perform any update or insert based upon the result, roll the dice again… and continue this loop until an exit condition is met.

Instead, consider creating an initial set of dice rolls that far exceeds the number needed and use set based windowing techniques to create your answer. In this scenario you will need to provide a validation to ensure you have a large enough set of dice rolls, but it will provide a more elegant solution that executes faster.

Conclusion

I hope you have enjoyed solving these puzzles as much as I have enjoyed creating them (and solving them myself).

For additional puzzles please visit my website at: **https://advancedsqlpuzzles.com/**

Happy Coding!

The End!