Advanced SQL Puzzles

Sequence, Selection, Iteration

Scott Peters

www.advancedsqlpuzzles.com

Last updated 03/14/2021

The answers provided at the end of the document were written in SQL Server T-SQL.

I would be happy to receive corrections, additions, new tricks and techniques, and other suggestions. scottpeters1188@outlook.com.

The latest version of this document can be found at www.advancedsqlpuzzles.com

---

Puzzle #1

Double or Add 1

Write a program where you start with 10 cents, and with each iteration, you can double your current amount or add 1 dollar. What is the smallest amount of iterations would it take to reach 1 million dollars?

Hyperlink to the solution

Puzzle #2

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, re-roll the dice and reduce your score by this amount. You cannot go lower than 0 points.

What was the least/greatest number of dice throws to reach 100 points?

Hyperlink to the solution

Puzzle #3

Jospehus Problem

Solve the Josephus Problem.

Once solved, any counting game (such as Eeny, Meeny, Miney, Moe) becomes quite simple.

Hyperlink to the solution

Puzzle #4

Non-Adjacent Numbers

Given the ordered set of numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; how many total arrangements of these numbers are possible where no two adjacent entries are adjacent numbers?

For example, the arrangement 1, 3, 5, 7, 9, 2, 4, 6, 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.

Hyperlink to the solution

Puzzle #5

Add the Numbers Up

Given the ordered numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, and a + or – sign anywhere between the digits; create all possible permutations and the amount in which they add up to.

Here are some examples:

1 + 2 - 3 + 4 - 567 + 8910 = 8347
12 + 3 - 4 + 56 - 789 = -722
1 + 2345678910 = 2345678911
1 - 2345678910 = -2345678909

Hyperlink to the solution

Puzzle #6

High-Low Card Game

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

The game is played by receiving a card and then determining if the next card will be of higher or lower value based upon probability. If the probability predicts the next card will be of the same value, or any of the probabilities of high, low or of the same value match, the computer must make a random decision between higher or lower.

Document an iteration through a deck of cards and if the probability matched the outcome.

Hyperlink to the solution

Puzzle #7

Pascal's Triangle

If you are unfamiliar with Pascal's Triangle, please review the Wikipedia page here.

For any row or position in Pascal's Triangle, can you compute the expected value?

Hyperlink to the solution

Puzzle #8

Permutations of 0 and 1

Can you display all permutations of the combination of 0 and 1 with a length of 6 characters?

Here are some examples:

000001,
101010,
001100,
111111,
000000,
000100,
011101, etc.…

Hyperlink to the solution

Puzzle #9

Permutations 1 through 10

Can you display all 2-digit permutations for the numbers 1 through 10?

Here are some examples:

1 and 2,
1 and 3,
1 and 4,
2 and 1,
2 and 3, etc…

Hyperlink to the solution

Answers

Answer to Puzzle #1

Double Or Add One

```
/*********************************************************************
Double Or Add One

Scott Peters
www.AdvancedSQLPuzzles.com

Developer Notes:

SQL Server's only type of loop is the WHILE loop, and you must combine this
with an IF statement to ensure you do not double your amount past the target amount.

*********************************************************************/

DECLARE @vAmountToMatch MONEY = 1000000;
DECLARE @vCurrentAmount MONEY = .01;
DECLARE @vIntegerterator INTEGER = 0;
DECLARE @vDifference INTEGER;

WHILE @vCurrentAmount <= @vAmountToMatch
       BEGIN
       --PRINT '@vIntegerterator is ' + CAST(@vIntegerterator AS VARCHAR)
       --PRINT '@vCurrentAmount is ' + CAST(@vCurrentAmount AS VARCHAR)
       IF     (@vAmountToMatch - @vCurrentAmount) > @vCurrentAmount
              SET @vCurrentAmount = @vCurrentAmount * 2;
       ELSE BREAK;

       SET    @vIntegerterator = @vIntegerterator + 1;
       END

SET @vIntegerterator = @vIntegerterator + (@vAmountToMatch - @vCurrentAmount);

PRINT 'The least number of iterations to reach ' + CAST(@vAmountToMatch AS VARCHAR) + '
is ' + CAST(@vIntegerterator AS VARCHAR);
```

Answer to Puzzle #2

Dice Throw Game

```
/*********************************************************************
Scott Peters
www.AdvancedSQLPuzzles.com

Developer Notes:

This is a fun puzzle to solve.

Two loops, one for the number of trials,
and one for the number of dice throws needed

Modify the following variable to change the number of trials
@vNumberOfTrials

*********************************************************************/


IF OBJECT_ID('tempdb.dbo.##Results','U') IS NOT NULL
  DROP TABLE ##Results;
GO

CREATE TABLE ##Results
(
IterationNumber INTEGER IDENTITY(1,1),
DiceThrowCount       INTEGER
);

--Modify the number of trials
DECLARE @vNumberOfTrials INTEGER = 1000

DECLARE @vNumberOfSteps INTEGER = 100;
DECLARE @vStepCount INTEGER;
DECLARE @vDiceThrow SMALLINT;
DECLARE @vDiceResult SMALLINT;
DECLARE @vDiceThrowCount SMALLINT;
DECLARE @vIntegerterationNumber INTEGER = 1

WHILE @vIntegerterationNumber <= @vNumberOfTrials
       BEGIN

       SET @vStepCount = 0;
       SET @vDiceThrowCount = 1;
       SET @vIntegerterationNumber = @vIntegerterationNumber + 1;

       WHILE @vStepCount < @vNumberOfSteps
              BEGIN

              SET @vDiceThrowCount =  @vDiceThrowCount + 1

              SELECT @vDiceResult = ABS(CHECKSUM(NEWID()) % 6) + 1

              IF @vDiceResult BETWEEN 1 AND 5
```

```
                    BEGIN
                    SET @vStepCount = @vStepCount + @vDiceResult
                    END

             IF @vDiceResult = 6
                    BEGIN
                    SELECT @vDiceResult = ABS(CHECKSUM(NEWID()) % 6) + 1
                    SET @vStepCount =
                    (CASE  WHEN @vStepCount - @vDiceResult < 0 THEN 0
                           ELSE @vStepCount - @vDiceResult END)
                    END

       END

INSERT INTO ##Results (DiceThrowCount)
SELECT @vDiceThrowCount;

END;


SELECT MAX(IterationNumber) as MaxIterationNumber,
       AVG(DiceThrowCount) AS AverageThrowsNeeded,
       MIN(DiceThrowCount) AS MinThrowsNeeded,
       MAX(DiceThrowCount) AS MaxThrowsNeeded,
       (
       SELECT TOP 1 DiceThrowCount
       FROM ##Results
       GROUP BY DiceThrowCount
       ORDER BY COUNT(*) DESC) AS Most_Occuring,
       (
       SELECT TOP 1 COUNT(*)
       FROM ##Results
       GROUP BY DiceThrowCount
       ORDER BY COUNT(*) DESC
       ) AS Most_Occuring_Count
FROM ##Results;

SELECT * FROM ##Results;
GO
```

Answer to Puzzle #3

Josephus Problem

```
/**************************************************************************************
Josephus Problem

Scott Peters
www.AdvancedSQLPuzzles.com

Developer Notes:

To learn more about the Josephus problem, please visit the Wikipedia page.
https://en.wikipedia.org/wiki/Josephus_problem

Note that I do not try to solve the Josephus problem by mathematical reasons,
but by performing the experiment round by round.

There are a few rules that I created for myself in solving the issue
* Do not use a DELETE statement
* I must document in which round the soldier was killed

Modify the following variables in the declarations section
to determine the soldier count and the number of soldiers to skip
       @vSoldierCount
       @vSoldiersToSkip

**************************************************************************************/

IF OBJECT_ID('tempdb.dbo.##RomanSoldiers','U') IS NOT NULL
  DROP TABLE ##RomanSoldiers;
GO

IF OBJECT_ID('tempdb.dbo.##RomanSoldiersTemp','U') IS NOT NULL
  DROP TABLE ##RomanSoldiersTemp;
GO

CREATE TABLE ##RomanSoldiers
(
--RowNumber INTEGER IDENTITY(1,1) NOT NULL,
RomanSoldierNumber INTEGER NOT NULL,
KillRound INTEGER NOT NULL DEFAULT 0,
);
GO

CREATE TABLE ##RomanSoldiersTemp
(
RowNumber INTEGER IDENTITY(1,1) NOT NULL,
RomanSoldierNumber INTEGER NOT NULL,
KillRound INTEGER NOT NULL,
);
GO
--End
------------------------------------------------------
------------------------------------------------------
PRINT 'Declare the variables';
```

```
DECLARE @vSoldierCount INTEGER = 10;
DECLARE @vSoldiersToSkip INTEGER = 6;
DECLARE @vSoldierToKill INTEGER;
DECLARE @vKillRound INTEGER = 1;
DECLARE @vRomanSoldierStartingPoint INTEGER;
DECLARE @vTableRecordCount INTEGER;
DECLARE @vInteger INTEGER = 1;

-----------------------------------
-----------------------------------
Print 'Populate the ##RomanSoldiers table'
WHILE @vInteger <= @vSoldierCount
       BEGIN
       INSERT INTO ##RomanSoldiers (RomanSoldierNumber) VALUES (@vInteger);
       SET @vInteger = @vInteger + 1
       END;
-----------------------------------

PRINT 'Enter While Statement';
WHILE (SELECT COUNT(*) FROM ##RomanSoldiers WHERE KillRound = 0) > 1

       BEGIN

       PRINT 'Truncate table ##RomanSoldiersTemp';
       TRUNCATE TABLE ##RomanSoldiersTemp;

       PRINT 'Determine @vRomanSoldierStartingPoint'
       SELECT @vRomanSoldierStartingPoint =
              (CASE WHEN @vKillRound = 1 THEN 1 ELSE MAX(RomanSoldierNumber) + 1 END)
       FROM   ##RomanSoldiers
       WHERE  KillRound <> 0

       PRINT 'The RomanSoldierStartingPoint is ' +
              CAST(@vRomanSoldierStartingPoint AS VARCHAR);

       IF @vRomanSoldierStartingPoint > @vSoldierCount
              BEGIN
              SELECT @vRomanSoldierStartingPoint = MIN(RomanSoldierNumber)
              FROM ##RomanSoldiers WHERE KillRound = 0;

              PRINT 'The RomanSoldierStartingPoint has been modified to ' +
                     CAST(@vRomanSoldierStartingPoint AS VARCHAR);
              END

       ---------------------------------
       --Populate the ##RomanSoldiersTemp
       PRINT 'Insert into ##RomanSoldiersTemp';
       INSERT INTO ##RomanSoldiersTemp (RomanSoldierNumber, KillRound)
       SELECT RomanSoldierNumber,
              KillRound
       FROM   ##RomanSoldiers
       WHERE  KillRound = 0 AND RomanSoldierNumber >= @vRomanSoldierStartingPoint
       ORDER BY RomanSoldierNumber;

       SELECT @vTableRecordCount = COUNT(*) FROM ##RomanSoldiersTemp;
       PRINT 'The @vTableRecordCount is ' + CAST(@vTableRecordCount AS VARCHAR);
```

```
	WHILE @vTableRecordCount < @vSoldiersToSkip + 1
		BEGIN
		PRINT 'Insert into ##RomanSoldiersTemp to increase record count';
		INSERT INTO ##RomanSoldiersTemp (RomanSoldierNumber, KillRound)
		SELECT RomanSoldierNumber,
			KillRound
		FROM   ##RomanSoldiers
		WHERE  KillRound = 0
		ORDER BY RomanSoldierNumber;

		SELECT @vTableRecordCount = COUNT(*) FROM ##RomanSoldiersTemp;
		PRINT 'The @vTableRecordCount has been modified to ' +
			CAST(@vTableRecordCount AS VARCHAR);
		END
	---------------------------------
	PRINT 'The @vKillRound number is ' + CAST(@vKillRound AS VARCHAR);

	SET @vSoldierToKill =
				(SELECT RomanSoldierNumber
				FROM ##RomanSoldiersTemp
				WHERE RowNumber = @vSoldiersToSkip + 1);
	PRINT 'The soldier to kill is ' + CAST(@vSoldierToKill AS VARCHAR);

	PRINT 'Update ##RomanSoldiers'
	UPDATE ##RomanSoldiers
	SET		KillRound = @vKillRound
	WHERE  RomanSoldierNumber = @vSoldierToKill;

	SET @vKillRound = @vKillRound + 1;
	PRINT 'Kill round has been incremented to ' + CAST(@vKillRound AS VARCHAR);


	END

SELECT a.*,
	(CASE KillRound WHEN 0 THEN 'Roman Soldier ' +
	CAST(RomanSoldierNumber AS VARCHAR) +
	' has been spared death!!' ELSE 'Dead!'  END) AS Note
FROM   ##RomanSoldiers a;
```

Answer to Puzzle #4

Non-Adjacent Numbers

```
/*********************************************************************
Permutations of Non-Adjacent Numbers

Scott Peters
www.AdvancedSQLPuzzles.com

Developer Notes:

The number of possible permutations for the numbers 1 through 10 with
no adjacent numbers is 424,807 out of a total possible 3,628,800 permutations.
This equates to 12% of the permutations do not have adjacent numbers.

This code takes approx. 3 minutes to run (on my fancy laptop of course).

*********************************************************************/
IF OBJECT_ID('tempdb.dbo.##InitialValues','U') IS NOT NULL
  DROP TABLE ##InitialValues;
GO

IF OBJECT_ID('tempdb.dbo.##Permutations','U') IS NOT NULL
  DROP TABLE ##Permutations;
GO


CREATE TABLE ##InitialValues
(
RowNumber INTEGER IDENTITY(1,1),
Element VARCHAR(100)
);
GO

INSERT INTO ##InitialValues (Element) VALUES
(1),(2),(3),(4),(5),(6),(7),(8),(9),(10);
GO


CREATE TABLE ##Permutations
(
RowNumber INTEGER IDENTITY(1,1) NOT NULL,
Permutation VARCHAR(1000) NOT NULL,
AdjacentNumbers BIT NULL
);
GO

DECLARE @vTotalElements INTEGER = (SELECT COUNT(*) FROM ##InitialValues);
DECLARE @vIntegerterator INTEGER = 1;
DECLARE @vElementValue VARCHAR(100);

--Populate the ##Permutations table with all possible permutations
WITH cte_Permutations (Permutation, Ids, Depth)
AS
(
SELECT CAST(Element AS VARCHAR(MAX)),
```

```
       CAST(RowNumber AS VARCHAR(MAX)) + ';',
       1 AS Depth
FROM   ##InitialValues
UNION ALL
SELECT a.Permutation + ',' + b.Element,
       a.Ids + CAST(b.RowNumber AS VARCHAR) + ';',
       a.Depth + 1
FROM   cte_Permutations a,
       ##InitialValues b
WHERE  a.Depth < @vTotalElements AND
       a.Ids NOT LIKE '%' + CAST(b.RowNumber AS VARCHAR) + ';%'
)
INSERT INTO ##Permutations (Permutation)
SELECT Permutation
FROM   cte_Permutations
WHERE  Depth = @vTotalElements;

------------------------------------------------------------
------------------------------------------------------------
--Determines permutations with adjacent numbers
WHILE @vIntegerterator <= @vTotalElements
       BEGIN

       SELECT @vElementValue = Element
       FROM ##InitialValues
       WHERE RowNumber = @vIntegerterator;

       PRINT '@vElementValue is ' + CAST(@vElementValue AS VARCHAR);

       UPDATE ##Permutations
       SET           AdjacentNumbers = 1
       WHERE  Permutation LIKE '%' +
              CAST(@vElementValue AS VARCHAR) + ',' +
              CAST(@vElementValue + 1 AS VARCHAR) + '%'
              OR
              Permutation LIKE '%' + CAST(@vElementValue AS VARCHAR) +
              ',' + CAST(@vElementValue -1 AS VARCHAR) + '%';

       SET @vIntegerterator = @vIntegerterator + 1;

       END

--View the results
SELECT * FROM ##Permutations WHERE AdjacentNumbers IS NULL;
```

Answer to Puzzle #5

Add The Numbers Up

```
/*********************************************************************
Scott Peters
www.AdvancedSQLPuzzles.com

Developer Notes:

This is the hardest puzzle to solve!

Runs in approximately 2 minutes.
19,683 different permutations.

I have 4 WHILE loops needed to complete the puzzle.

To sum the equation, I use dynamic SQL and a RBAR approach.
If anyone can do this as a set operation, please contact me!

*********************************************************************/


IF OBJECT_ID('tempdb.dbo.##InitialValues','U') IS NOT NULL
  DROP TABLE ##InitialValues;
GO

IF OBJECT_ID('tempdb.dbo.##PossibleValues','U') IS NOT NULL
  DROP TABLE ##PossibleValues;
GO

IF OBJECT_ID('tempdb.dbo.##Equations','U') IS NOT NULL
  DROP TABLE ##Equations;
GO

IF OBJECT_ID('tempdb.dbo.##EquationsFinal','U') IS NOT NULL
  DROP TABLE ##EquationsFinal;
GO

CREATE TABLE ##InitialValues
(
RowNumber INTEGER IDENTITY(1,1),
Element VARCHAR(100)
);
GO

INSERT INTO ##InitialValues (Element) VALUES
(1),(2),(3),(4),(5),(6),(7),(8),(9),(10);
GO

CREATE TABLE ##PossibleValues
(
RowNumber INTEGER IDENTITY(1,1) NOT NULL,
ElementOriginal INTEGER NOT NULL,
ElementAdded INTEGER NOT NULL,
ElementModified BIGINT NOT NULL
```

```
);
GO

CREATE TABLE ##Equations
(
RowNumber INTEGER IDENTITY(1,1) NOT NULL,
InsertIterator INTEGER,
ElementOriginal INTEGER,
ElementOriginal2 INTEGER,
ElementAdded bigint,
ElementAddedLastNumber BIGINT,
ElementModifiedVarchar VARCHAR(1000)
);
GO

CREATE TABLE ##EquationsFinal
(
RowNumber INTEGER IDENTITY(1,1) NOT NULL,
ElementModifiedVarchar VARCHAR(1000),
TotalSumInteger BIGINT,
);
GO

--Declare your variables
DECLARE @vIntegerterator1 INTEGER = 1;
DECLARE @vIntegerterator2 INTEGER;
DECLARE @vIntegerterator3 INTEGER = 2;
DECLARE @vElementValue VARCHAR(100);
DECLARE @vElementValueOriginal VARCHAR(100);
DECLARE @vElementValueAdded VARCHAR(100);
DECLARE @vElementValueModified VARCHAR(100);


--Populates the ##PossibleValues table
--WHILE loop within a WHILE loop
PRINT 'Entering While Loop 1'
WHILE @vIntegerterator1 <= (SELECT COUNT(*) FROM ##InitialValues)
       BEGIN
       SELECT @vElementValueOriginal = Element FROM ##InitialValues WHERE RowNumber =
@vIntegerterator1;
       PRINT @vElementValue
       INSERT INTO ##PossibleValues(ElementOriginal,ElementAdded,ElementModified)
       VALUES(@vElementValueOriginal,@vElementValueOriginal,@vElementValueOriginal)
       SET @vElementValueModified = @vElementValueOriginal

       SET @vIntegerterator2 = @vIntegerterator1
       PRINT 'Entering While Loop 2'
       WHILE @vIntegerterator2 < (SELECT COUNT(*) FROM ##InitialValues)
              BEGIN
              SELECT @vElementValueAdded = Element
              FROM ##InitialValues
              WHERE RowNumber = @vIntegerterator2 + 1;
              SET @vElementValueModified =
                     CONCAT(@vElementValueModified,@vElementValueAdded)
              PRINT @vElementValueModified
              INSERT INTO ##PossibleValues(ElementOriginal,ElementAdded,ElementModified)
              VALUES (@vElementValueOriginal, @vElementValueAdded,@vElementValueModified)
```

```
              SET    @vIntegerterator2 = @vIntegerterator2 + 1
              END;
       PRINT 'End While Loop 2'
SET    @vIntegerterator1 = @vIntegerterator1 + 1
END


--Seeds The ##Equations table
INSERT INTO ##Equations
(InsertIterator,ElementOriginal,ElementOriginal2,ElementAdded,ElementAddedLastNumber,Elem
entModifiedVarchar)
SELECT 1,
       ElementOriginal,
       ElementOriginal,
       ElementModified,
       ElementAdded,
       CAST(ElementModified AS VARCHAR) AS ElementModifiedVarchar
FROM   ##PossibleValues
WHERE  ElementOriginal = '1';


--Builds The ##Equations table Of all possible permuations
WHILE @vIntegerterator3 <= (SELECT COUNT(*) FROM ##InitialValues)
       BEGIN
       INSERT INTO ##Equations
       (InsertIterator,ElementOriginal,ElementOriginal2,ElementAdded,
       ElementAddedLastNumber,ElementModifiedVarchar)
       SELECT @vIntegerterator3 AS InsertIterator,
              a.ElementOriginal AS ElementOriginal,
              b.ElementOriginal AS ElementOriginal2,
              b.ElementModified AS ElementAdded,
              b.ElementAdded AS ElementAddedLastNumber,
              CONCAT(a.ElementModifiedVarchar,'+', CAST(b.ElementModified AS VARCHAR))
       FROM   ##Equations a INNER JOIN
              ##PossibleValues b ON a.ElementAddedLastNumber + 1 = b.ElementOriginal
       WHERE  a.InsertIterator = @vIntegerterator3 - 1
       UNION
       SELECT @vIntegerterator3,
              a.ElementOriginal,
              b.ElementOriginal AS ElementOriginal2,
              b.ElementModified AS ElementAdded,
              b.ElementAdded,
              CONCAT(a.ElementModifiedVarchar,'-', CAST(b.ElementModified AS VARCHAR))
       FROM   ##Equations a INNER JOIN
              ##PossibleValues b ON a.ElementAddedLastNumber + 1 = b.ElementOriginal
       WHERE  a.InsertIterator = @vIntegerterator3 - 1;


       SET @vIntegerterator3 = @vIntegerterator3 + 1;
       END;



--Populates The ##EquationsFinal table
INSERT INTO ##EquationsFinal (ElementModifiedVarchar)
SELECT ElementModifiedVarchar
FROM   ##Equations
```

```
WHERE  ElementModifiedVarchar LIKE '%' +
       CAST((SELECT MAX(CAST(Element AS INTEGER)) FROM ##InitialValues) AS VARCHAR) + ''
GO

----------------------------------------
--Add the Numbers Up
--This processes goes row by row to add the numbers up
--Updates the TotalSum field in the ##EquationsFinal table
DECLARE @vRowNumber INTEGER = 1;
DECLARE @vEquation VARCHAR(1000);
DECLARE @vSum BIGINT;
DECLARE @vSQLStatement NVARCHAR(1000);

WHILE @vRowNumber <= (SELECT COUNT(*) FROM ##EquationsFinal)
       BEGIN
       SELECT @vEquation = ElementModifiedVarchar
       FROM ##EquationsFinal
       WHERE RowNumber = @vRowNumber;
       PRINT '@vEquation is ' + @vEquation

       SELECT @vSQLStatement = 'SELECT @var = ' + @vEquation;
       EXECUTE sp_executesql @vSQLStatement, N'@var BIGINT OUTPUT', @var = @vSum OUTPUT
       PRINT '@vSum is ' + CAST(@vSum AS VARCHAR(100));

       UPDATE ##EquationsFinal
       SET           TotalSumInteger = @vSum
       WHERE  RowNumber = @vRowNumber;

       SET @vRowNumber = @vRowNumber + 1
       PRINT '@vRowNumber is ' + CAST(@vRowNumber AS VARCHAR);
       END


SELECT * FROM ##EquationsFinal;
```

Answer to Puzzle #6

High Low Card Game

```
/*************************************************************************
High Low Card Game

Scott Peters
www.AdvancedSQLPuzzles.com

Developer Notes:

Here are the steps I use to solve the problem.

1) INSERT a deck of cards into table ##CardDeck
2) Shuffle the cards and INSERT the cards into ##CardDeckSchuffled
3) Draw a card and INSERT this card into ##CardDeckPlayed
4) Predict the outcome of the value of the next card and UPDATE the ##CardDeckPlayed with
the prediction
5) Draw the next card in the deck from the table ##CardDeckSchuffled
6) UPDATE the ##CardDeckPlayed with the outcome of the prediction

*************************************************************************/

IF OBJECT_ID('tempdb.dbo.##CardDeck','U') IS NOT NULL
  DROP TABLE ##CardDeck;
GO

IF OBJECT_ID('tempdb.dbo.##CardDeckShuffled','U') IS NOT NULL
  DROP TABLE ##CardDeckShuffled;
GO

IF OBJECT_ID('tempdb.dbo.##CardDeckPlayed','U') IS NOT NULL
  DROP TABLE ##CardDeckPlayed;
GO

CREATE TABLE ##CardDeck
(
RandomNumber UNIQUEIDENTIFIER DEFAULT NEWID(),
CardValue INTEGER,
Suit VARCHAR(10),
CardDescription VARCHAR(10)
);
GO

INSERT INTO ##CardDeck (CardValue, Suit, CardDescription)
VALUES
(2,'Spades','Two'),
(3,'Spades','Three'),
(4,'Spades','Four'),
(5,'Spades','Five'),
(6,'Spades','Six'),
(7,'Spades','Seven'),
(8,'Spades','Eight'),
(9,'Spades','Nine'),
(10,'Spades','Ten'),
```

```
(11,'Spades','Jack'),
(12,'Spades','Queen'),
(13,'Spades','King'),
(14,'Spades','Ace'),
(2,'Clubs','Two'),
(3,'Clubs','Three'),
(4,'Clubs','Four'),
(5,'Clubs','Five'),
(6,'Clubs','Six'),
(7,'Clubs','Seven'),
(8,'Clubs','Eight'),
(9,'Clubs','Nine'),
(10,'Clubs','Ten'),
(11,'Clubs','Jack'),
(12,'Clubs','Queen'),
(13,'Clubs','King'),
(14,'Clubs','Ace'),
(2,'Hearts','Two'),
(3,'Hearts','Three'),
(4,'Hearts','Four'),
(5,'Hearts','Five'),
(6,'Hearts','Six'),
(7,'Hearts','Seven'),
(8,'Hearts','Eight'),
(9,'Hearts','Nine'),
(10,'Hearts','Ten'),
(11,'Hearts','Jack'),
(12,'Hearts','Queen'),
(13,'Hearts','King'),
(14,'Hearts','Ace'),
(2,'Diamonds','Two'),
(3,'Diamonds','Three'),
(4,'Diamonds','Four'),
(5,'Diamonds','Five'),
(6,'Diamonds','Six'),
(7,'Diamonds','Seven'),
(8,'Diamonds','Eight'),
(9,'Diamonds','Nine'),
(10,'Diamonds','Ten'),
(11,'Diamonds','Jack'),
(12,'Diamonds','Queen'),
(13,'Diamonds','King'),
(14,'Diamonds','Ace');
GO

CREATE TABLE ##CardDeckPlayed
(
RowNumber INTEGER,
CardValue INTEGER,
Suit VARCHAR(10),
CardDescription VARCHAR(10),
PercentNextCardIsHigher FLOAT,
PercentNextCardIsLower FLOAT,
PercentNextCardIsSame FLOAT,
NextCardValue INTEGER,
Prediction VARCHAR(1000),
Outcome BIT
```

```
);
GO

DECLARE @vCurrentCardValue INTEGER;
DECLARE @vRowNumber INTEGER = 1;

DECLARE @vTotalCardsPlayed FLOAT;
DECLARE @vTotalCardsHigher FLOAT;
DECLARE @vTotalCardsLower FLOAT;
DECLARE @vTotalCardsSame FLOAT = 4;
DECLARE @vTotalCardsPlayedHigher FLOAT;
DECLARE @vTotalCardsPlayedLower FLOAT;
DECLARE @vTotalCardsPlayedSame FLOAT;
DECLARE @vPercentHigher FLOAT;
DECLARE @vPercentLower FLOAT;
DECLARE @vPercentSame FLOAT;

DECLARE @vLastPlayedCard INTEGER /*Used in the print statement*/

--Schuffle the cards
SELECT Row_Number() OVER (ORDER BY RandomNumber) as RowNumber,
       a.*
INTO   ##CardDeckShuffled
FROM   ##CardDeck a
ORDER BY 1;


--Begin Loop
WHILE @vRowNumber <= 52
       BEGIN

       --Draw a card and insert into table
       INSERT INTO ##CardDeckPlayed (RowNumber, CardValue, Suit, CardDescription)
       SELECT RowNumber,
              CardValue,
              Suit,
              CardDescription
       FROM   ##CardDeckShuffled
       WHERE  RowNumber = @vRowNumber;

       --If its the last card, exit the predictions
       IF @vRowNumber = 52 BREAK

       --Read the card
       SELECT @vCurrentCardValue = CardValue,
              @vRowNumber = RowNumber
       FROM   ##CardDeckPlayed
       WHERE  RowNumber = @vRowNumber;

       --Determine prediction
       SELECT @vTotalCardsPlayed = COUNT(*)
       FROM ##CardDeckPlayed;

       SELECT @vTotalCardsHigher = COUNT(*)
       FROM ##CardDeckShuffled WHERE CardValue > @vCurrentCardValue;

       SELECT @vTotalCardsLower = COUNT(*)
```

```
FROM ##CardDeckShuffled WHERE CardValue < @vCurrentCardValue;

SELECT @vTotalCardsPlayedHigher = COUNT(*)
FROM ##CardDeckPlayed WHERE CardValue > @vCurrentCardValue;

SELECT @vTotalCardsPlayedLower = COUNT(*)
FROM ##CardDeckPlayed WHERE CardValue < @vCurrentCardValue;

SELECT @vTotalCardsPlayedSame = COUNT(*)
FROM ##CardDeckPlayed WHERE CardValue = @vCurrentCardValue;


--Percentage that the next card is higher
SET @vPercentHigher =
(@vTotalCardsHigher - @vTotalCardsPlayedHigher) / (52 - @vTotalCardsPlayed);

--Percentage that the next card is lower
SET @vPercentLower =
(@vTotalCardsLower - @vTotalCardsPlayedLower) / (52 - @vTotalCardsPlayed);

--Percentage that the next card is same
SET @vPercentSame =
(@vTotalCardsSame - @vTotalCardsPlayedSame) / (52 - @vTotalCardsPlayed);

--Update table with percentages
UPDATE ##CardDeckPlayed
SET    PercentNextCardIsHigher = @vPercentHigher,
       PercentNextCardisLower = @vPercentLower,
       PercentNextCardisSame = @vPercentSame
WHERE  RowNumber = @vRowNumber;

--Increase the iterator
SET @vRowNumber = @vRowNumber + 1;

--Read the next card
SELECT @vCurrentCardValue = CardValue,
              @vRowNumber = RowNumber
FROM   ##CardDeckShuffled
WHERE  RowNumber = @vRowNumber;

--Determine the prediciton
UPDATE ##CardDeckPlayed
SET    NextCardValue = @vCurrentCardValue,
       Prediction =
              CASE
              WHEN   @vPercentHigher > @vPercentLower AND
                     @vPercentHigher > @vPercentSame THEN 'Prediciton - Higher'
              WHEN   @vPercentLower > @vPercentHigher AND
                     @vPercentLower > @vPercentSame THEN 'Prediciton - Lower'
              WHEN   --Scenario 1 - Two matching predicitons
                     @vPercentHigher = @vPercentLower OR
              --Scenario 2 - The prediction of same is the highest prediction
                     @vPercentSame > @vPercentHigher AND
                     @vPercentSame > @vPercentLower
              THEN   (CASE
                     WHEN SUBSTRING(CAST(RAND() AS VARCHAR),3,1)
                     IN ('0','1','2','3','4')
```

```
                        THEN 'Guessing - Higher' ELSE 'Guessing - Lower' END) END

      WHERE  RowNumber = @vRowNumber - 1;--Remmeber i increased the incrementer above

      --Was the prediction correct?
      UPDATE ##CardDeckPlayed
      SET    Outcome =
                  CASE
                  WHEN   Prediction LIKE '%Higher%' AND CardValue < NextCardValue
                  THEN 1
                  WHEN   Prediction LIKE '%Lower%' AND CardValue > NextCardValue
                  THEN 1
                  ELSE   0 END
      WHERE  RowNumber = @vRowNumber - 1;

      END;

SELECT * FROM ##CardDeckPlayed;
```

Answer to Puzzle #7

Pascal's Triangle

```
/*********************************************************************
Pascal's Triangle

Scott Peters
www.AdvancedSQLPuzzles.com

Developer Notes:

Note the equation uses 0 based indexing.
The first row is row 0 and not row 1.
The first position is position 0 and not position 1.

Also, the factorial of 0! is 1.

Here is the first eight rows.  Remember the equation uses zero based indexing.
1             1 |
2             1 | 1 |
3             1 | 2 |  1 |
4             1 | 3 |  3 |  1 |
5             1 | 4 |  6 |  4 |  1 |
6             1 | 5 | 10 | 10 |  5 |  1 |
7             1 | 6 | 15 | 20 | 15 |  6 | 1 |
8             1 | 7 | 21 | 35 | 35 | 21 | 7 | 1 |

Pos.   1 | 2 |  3 |  4 |  5 |  6 | 7 | 8 |

Modify the following variables to determine the row and position
@vRowInput
@vPositionInput

*********************************************************************/

--Set your row and position.
--This does not use 0 based indexing.  I account for this below.
DECLARE @vRowInput INTEGER = 7;
DECLARE @vPositionInput INTEGER = 4;



PRINT 'Declare Variables'
------------
DECLARE @vRowNumber INTEGER = @vRowInput - 1;
DECLARE @vPositionNumber INTEGER = @vPositionInput - 1;
DECLARE @vPositionValue INTEGER;
------------
--Row
DECLARE @vRowFactorial INTEGER;
DECLARE @vRowFactorialIterator INTEGER = 1;
------------
--Position
DECLARE @vPositionFactorial INTEGER;
DECLARE @vPositionFactorialIterator INTEGER = 1;
```

```
------------
--Row Minus Position
DECLARE @vRowMinusPositionFactorial INTEGER;
DECLARE @vRowMinusPositionFactorialIterator INTEGER = 1;
DECLARE @vRowMinusPositionNumber INTEGER;

PRINT '--------------------'
PRINT 'The @vRowNumber is ' + CAST(@vRowNumber AS VARCHAR);
PRINT 'The @vPositionNumber is ' + CAST(@vPositionNumber AS VARCHAR);
PRINT '--------------------'
------------

IF @vRowNumber = 0 OR @vPositionNumber = 0
       BEGIN
       PRINT 'Setting @vPositionValue = 1'
       SET @vPositionValue = 1
       RETURN;
       END;


------------
PRINT 'Determine Row Factorial'
SET @vRowFactorial = @vRowNumber;

WHILE @vRowFactorialIterator < @vRowNumber
       BEGIN
       --PRINT '@vRowFactorial is ' + CAST(@vRowFactorial AS VARCHAR);
       --PRINT '@vRowFactorialIterator is ' + CAST(@vRowFactorialIterator AS VARCHAR);

       SET @vRowFactorial = @vRowFactorial * @vRowFactorialIterator;
       SET @vRowFactorialIterator = @vRowFactorialIterator + 1
       END
PRINT '@vRowFactorial is ' + CAST(@vRowFactorial AS VARCHAR);
PRINT '-----------------------'


------------
PRINT 'Determine Position Factorial'
SET @vPositionFactorial = @vPositionNumber;

WHILE @vPositionFactorialIterator < @vPositionNumber
       BEGIN
       --PRINT '@vPositionFactorial is ' + CAST(@vPositionFactorial AS VARCHAR);
       --PRINT '@vPositionFactorialIterator is ' + CAST(@vPositionFactorialIterator AS
VARCHAR);
       SET @vPositionFactorial = @vPositionFactorial * @vPositionFactorialIterator;
       SET @vPositionFactorialIterator = @vPositionFactorialIterator + 1
       END
PRINT '@vPositionFactorial is ' + CAST(@vPositionFactorial AS VARCHAR);
PRINT '-----------------------'

------------
PRINT 'Determine Row Minus Position Factorial'
SET @vRowMinusPositionFactorial = (CASE WHEN @vRowNumber - @vPositionNumber = 0 THEN 1
ELSE @vRowNumber - @vPositionNumber END);
SET @vRowMinusPositionNumber = @vRowMinusPositionFactorial;
```

```
PRINT '@vRowMinusPositionFactorial is ' + CAST(@vRowMinusPositionFactorial AS VARCHAR)
WHILE @vRowMinusPositionFactorialIterator < @vRowMinusPositionNumber
              BEGIN
              SET @vRowMinusPositionFactorial =
                     @vRowMinusPositionFactorial * @vRowMinusPositionFactorialIterator;
              SET @vRowMinusPositionFactorialIterator =
                     @vRowMinusPositionFactorialIterator + 1
              PRINT @vRowMinusPositionFactorial;
              END

PRINT '--------------------'
PRINT @vPositionValue;
PRINT  'The value for Row ' + CAST(@vRowInput AS VARCHAR(100)) +
       ' and Position ' + CAST(@vPositionInput AS VARCHAR(100)) + '
       is ' + CAST(@vRowFactorial / (@vPositionFactorial *
              @vRowMinusPositionFactorial) AS VARCHAR);
```

Answer to Puzzle #8

Permutations Of 0 and 1

```
/*********************************************************************
Permutations Of 0 and 1

Scott Peters
www.AdvancedSQLPuzzles.com

Developer Notes:

This may be able to be solved via a single declarative statement using recursion.
However, it is very simple to solve with a single loop.

*********************************************************************/


IF OBJECT_ID('tempdb.dbo.##Permutations','U') IS NOT NULL
  DROP TABLE ##Permutations;
GO

CREATE TABLE ##Permutations
(
Permutation varchar(max)
);
GO

INSERT INTO ##Permutations (Permutation) VALUES ('0'),('1');
GO

--Modify this variable with the length of the string you want.
DECLARE @vPermutationLength INTEGER = 6

WHILE (SELECT MAX(LEN(Permutation)) FROM ##Permutations) <= @vPermutationLength
       BEGIN

       INSERT INTO ##Permutations (Permutation)
       SELECT CONCAT(a.Permutation,b.Permutation)
       FROM   ##Permutations a CROSS JOIN
              ##Permutations b;

       END;


SELECT DISTINCT
       LEFT(Permutation, @vPermutationLength) AS ZeroAndOne
FROM   ##Permutations
WHERE  LEN(Permutation) = @vPermutationLength;
```

Answer to Puzzle #9

Permutations 1 Through 10

```
/*********************************************************************
Permutations Of 1 Through 10

Scott Peters
www.AdvancedSQLPuzzles.com

Developer Notes:

This may be able to be solved via a single declarative statement using recursion.
However, it is very simple to solve with a single loop.

*********************************************************************/

IF OBJECT_ID('tempdb.dbo.##Values','U') IS NOT NULL
  DROP TABLE ##Values;
GO

IF OBJECT_ID('tempdb.dbo.##Permutations','U') IS NOT NULL
  DROP TABLE ##Permutations;
GO



CREATE TABLE ##Values
(
RowNumber INTEGER IDENTITY(1,1) NOT NULL,
MyValue VARCHAR(2)
);

CREATE TABLE ##Permutations
(
Permutation VARCHAR(10),
NumericDiffernece INTEGER
);
GO

INSERT INTO ##Values (MyValue)
VALUES ('1'),('2'),('3'),('4'),('5'),('6'),('7'),('8'),('9'),('10');
GO

DECLARE @vRowNumber INTEGER = 1;


WHILE @vRowNumber <= (SELECT MAX(RowNumber) FROM ##Values)
       BEGIN
       WITH cte_RowNumber AS
       (
       SELECT *
       FROM   ##Values
       WHERE  RowNumber = @vRowNumber
       )
       INSERT INTO ##Permutations (Permutation, NumericDiffernece)
```

```
        SELECT a.MyValue + ', ' + b.MyValue,
                    ABS(CAST(a.MyValue AS INTEGER) - CAST(b.MyValue AS INTEGER))
        FROM   cte_RowNumber a CROSS JOIN
                    ##VALUES b
        WHERE  a.MyValue <> b.MyValue;

        SET @vRowNumber = @vRowNumber + 1;

        END

SELECT * FROM ##Permutations;
```