Data Profiling with T-SQL

sqlservercentral.com/articles/data-profiling-with-t-sql

Adam Aspin

“Know yourself” is a famous ancient Greek axiom. “Know your data” is today’s variant, and it applies particularly well to DBAs and developers. Whether you are inheriting a database or developing an ETL process you will need to have a reasonably precise idea of the structure and content of the data you are dealing with. Amongst other elements, it can really help to know the following:

- The data type for each column in each table;
- The real data type for the data held in a column (does a VARCHAR column only contain INT data, for instance);
- The maximum and minimum lengths of (N)VARCHAR fields

It can also help to have a fair idea of the kind of data contained in certain fields, this can include:

- The number of NULLs;
- The percentage of NULLs;
- The number of zero-length fields;
- The percentage of zero-length fields;

Finally it can be useful to get an idea of the distribution of data in a field, if only to get an idea of the data contained in a field without having to scroll down through thousands of records. Admittedly this is not always essential, and its usefulness will depend on the data being analysed. But as the whole point of this exercise is to know the data, let us presume for the moment that it can be helpful to see how many and what percentage of a field is represented by each element in the recordset.

The next question is probably to ask why this information can be useful. I will always maintain that appropriate data types and data lengths make many data processes (from data loads to queries) run faster. You will save on disk space and backup times. Equally, elementary data validation can only save time spent analysing apparent errors further down the line. Also building SSIS packages is infinitely easier if the data types are appropriate and accurate.

When should you profile your data?

Clearly there are key stages in the development cycle when profiling the data can be important. These include (amongst others):

When creating a data import process

If you are loading what is essentially someone else's data into your database then you are trusting the external source to a large extent. Without advocating paranoia as a way of life, I strongly advise that you verify the type and content of external data at an early stage of the ETL process.

Imagine that you are importing a multi-million row table from an external RDBMS (SQL Server or not). You will probably be using a data dictionary provided by the third party to define the field types for your destination database. If no data dictionary is available then you could be using SSIS to deduce the data types. In any case you are being forced to trust the metadata definition of a third party. Can you be sure that this definition is exact? Are you certain that the developer was able to work to a precise specification? Were they in a hurry, and so set an NVARCHAR(3000) while telling themselves that they would switch this to an NVARCHAR(25) at a later date? Then they were overwhelmed by other more urgent tasks, until a temporary fix became a permanent feature. You probably get the idea by now.

Why does this matter, you may ask? Speed is the simple answer. When loading data when using SSIS, it is important to fit as many rows into the SSIS pipeline buffers as possible. The narrower the data type, the more records will be loaded into each buffer and the faster the load will finish. Remember: SSIS uses the maximum possible data length provided by the source metadata to calculate row widths, not the width of the actual field in each record.

It goes without saying that using the correct data types is fundamental. Not only will this decrease load times, but it will decrease processing times at a later stage in the ETL cycle and avoid pointless datatype conversions.

However, once the basics are dealt with, there are other things to consider. At the level of the data contained in a column, even if the competence of the original developer is not in doubt, what about the rigour of the people entering the data? Can you be sure that a field which should never allow NULLs was set to allow them – and that you have a disturbingly large percentage of NULLs which, in effect, invalidates the data set? In this case it is better to know early so you can bounce back your doubts to the data supplier before spending hours developing a process which is destined to be binned.

At another level, let's suppose that you are ingesting data using the "Extract, Load, Transform" paradigm. This means that you will be taking source data and loading into wide character columns from a data source, and subsequently converting the data into an appropriate data type. This will mean, initially, deducing the appropriate data type. Profiling the source data independently of the table metadata is fundamental to let you deduce and define the most accurate possible data type for each final, validated column in the definitive data table.

When inheriting a database

Many of the reasons given in the previous section apply equally well to inherited databases. Here, however, the suitable datatype is necessary not to accelerate data loads, but to reduce processing times and space used on disk. Yes, I know, compression is now used in many databases, disk space is cheap and…

Call me a purist, but I prefer appropriate data types and widths. In any case, how many inappropriate query plans are being used due to implicit conversions in the queries being used in the databases which you have inherited? If the answer is “none at all” then you don’t have a problem. But a more detailed analysis of query plans might just return another answer: “oops…too many”.

A (fairly) simple data profiling script

Assuming, then, that you accept the premise that profiling your data can be a good idea, here is a simple way to carry out basic data profiling on SQL Server tables. This script will run on SQL Server versions 2008 and above. The script is designed to profile a single table, and what it does is to:

- Get the core metadata for the source table (column name, datatype and length);
- Define a temporary tabe structure to hold the required profiling data;
- Process each column to return the profiling data;
- Detect any foreign keys;
- Perform any calculations of ratios/percentages;
- Add any suggestions for potential changes;
- Perform elementary domain analysis (number of column values at upper length thresholds, number of distinct elements in a column and the percentage of the total that this represents).
- Detect any foreign keys;

The contents of the resulting temporary tables are then returned.

Inevitably, the profile metadata which I have suggested above is only a subset of the range of profile data, which can be useful when analysing a source data set. However, rather than attempt to let the best be the enemy of the good, I prefer to suggest a script that will return a core set of profile metadata, which I have found to be extremely useful as a starting point. You can then, of course, extend the script to add any other elements that you find useful. A few suggestions which I have not had the time to code (yet) are given at the end of this article.

Notes on the script

I am using the INFORMATION_SCHEMA tables to source table and column metadata. You can use the system views if you prefer. I have added minimal error trapping, plus a few elementary checks and balances exist in the script to ensure that :

- The table exists;
- The selected columns exist;

The script expects a few input variables. These are:

- The table schema (this is required);
- The table name (this is required);
- An optional comma-delimited list of columns to profile (if you leave this blank then all columns are profiled);

There are also a series of predefined flags (all set to true) which are:

- A flag set to indicate if you want text columns profiling;
- A flag set to indicate if you want numeric columns profiling;
- A flag set to indicate if you want date columns profiling;
- A flag set to indicate if you want Large Object column types columns profiling;
- A flag set to indicate if you want domain analysis applied (this can take some time);

Finally there are

- The threshold at which a reference table or domain analysis is performed;
- The percentage of records at the upper or lower threshold which suggests a possible anomaly;
- The percentage of NULLs in a column which is used to suggest a possible anomaly;
- The percentage variance allowed from a defined data type used when suggesting another data type for a column;

Please note that the suggestions returned by the script as to potential data type modifications are just that – suggestions. You do not have to take them at face value. Indeed, you should thoroughly test any changes which you propose making on your data tables in a development environment and ensure that any existing processes that use that data will accept the changes which you make. Inevitably you will need to be as confident as possible that the source data will not change to preclude the use of a suggested data type.

I have to warn you that this script can take an extremely long tie to run on tables that are extremely wide (I have run it against tables of 400+ columns). Also – and inevitably – profiling the data in tables of tens of millions of records will take time. It can be advantageous to start by profiling only core elements (by setting some or all of the optional flags to 0) to get an initial take on your dataset. Then you can run further profiles on subsets of columns to analyse the areas which seem of particular interest. For instance, analysing the distribution of a unique ID is pointless.

Here is the script which you can run to profile a SQL Server table:

```
/*
This script is given "As Is" with no warranties and plenty of caveats. Use at your
own risk!
For more on data profiling, see Chapter 10 in "SQL Server 2012 Data Integration
Recipes", Apress, 2012
*/
-----------------------------------------------------------------------
-- User-defined variables
-----------------------------------------------------------------------
USE CarSales -- Your database here
GO
DECLARE @TABLE_SCHEMA NVARCHAR(128) = 'dbo'  -- Your schema here
DECLARE @TABLE_NAME NVARCHAR(128) = 'client' -- Your table here
DECLARE @ColumnListIN NVARCHAR(4000) = ''    -- Enter a comma-separated list of
specific columns
                                                    -- to profile, or leave blank
for all
DECLARE @TextCol BIT = 1  -- Analyse all text (char/varchar/nvarchar) data type
columns
DECLARE @NumCol BIT = 1   -- Analyse all numeric data type columns
DECLARE @DateCol BIT = 1  -- Analyse all date data type data type columns
DECLARE @LobCol BIT = 1   -- Analyse all VAR(char/nchar/binary) MAX data type
columns (potentially time-consuming)
DECLARE @AdvancedAnalysis BIT = 1 -- Perform advanced analysis (threshold
counts/domain analysis)
                                  --(potentially time-consuming)
DECLARE @DistinctValuesMinimum INT = 200 -- Minimum number of distinct values to
suggest a reference
                                       -- table and/or perform domain analysis
DECLARE @BoundaryPercent NUMERIC(3,2) = 0.57 -- Percent of records at upper/lower
threshold to suggest
                                           -- a possible anomaly
DECLARE @NullBoundaryPercent NUMERIC(5,2) = 90.00 -- Percent of NULLs to suggest a
possible anomaly
DECLARE @DataTypePercentage INT = 2 -- Percentage variance allowed when suggesting
another data type
                                    -- for a column
-----------------------------------------------------------------------
-- Process variables
-----------------------------------------------------------------------
DECLARE @DATA_TYPE VARCHAR(128) = ''
DECLARE @FULLSQL VARCHAR(MAX) = ''
DECLARE @SQLMETADATA VARCHAR(MAX) = ''
DECLARE @NUMSQL VARCHAR(MAX) = ''
DECLARE @DATESQL VARCHAR(MAX) = ''
DECLARE @LOBSQL VARCHAR(MAX) = ''
DECLARE @COLUMN_NAME VARCHAR(128)
DECLARE @CHARACTER_MAXIMUM_LENGTH INT
DECLARE @ROWCOUNT BIGINT = 0
DECLARE @ColumnList VARCHAR(4000) = ' '
DECLARE @TableCheck TINYINT
DECLARE @ColumnCheck SMALLINT
DECLARE @DataTypeVariance INT
-----------------------------------------------------------------------
-- Start the process:
BEGIN
TRY
-- Test that the schema and table exist
SELECT
```

```
 @TableCheck = COUNT (*)
   FROM INFORMATION_SCHEMA.TABLES
   WHERE TABLE_SCHEMA = @TABLE_SCHEMA
   AND TABLE_NAME = @TABLE_NAME
IF @TableCheck <> 1
 BEGIN
  RAISERROR ('The table does not exist',16,1)
  RETURN
 END
-----------------------------------------------------------------------
-- Parse list of columns to process / get list of columns according to types
required
-----------------------------------------------------------------------
IF OBJECT_ID('tempdb..#ColumnList') IS NOT NULL
 DROP TABLE tempdb..#ColumnList;
CREATE TABLE #ColumnList (COLUMN_NAME VARCHAR(128), DATA_TYPE VARCHAR(128),
CHARACTER_MAXIMUM_LENGTH INT) -- Used to hold list of columns to process
IF @ColumnListIN <> '' -- See if there is a list of columns to process
BEGIN
 -- Process list
 SET @ColumnList = @ColumnListIN + ','
 DECLARE @CharPosition int
 WHILE CHARINDEX(',', @ColumnList) > 0
  BEGIN
   SET @CharPosition = CHARINDEX(',', @ColumnList)
   INSERT INTO #ColumnList (COLUMN_NAME) VALUES (LTRIM(RTRIM(LEFT(@ColumnList,
@CharPosition - 1))))
   SET @ColumnList = STUFF(@ColumnList, 1, @CharPosition, '')
  END -- While loop
-- update with datatype and length
  UPDATE CL
   SET CL.CHARACTER_MAXIMUM_LENGTH = ISNULL(ISC.CHARACTER_MAXIMUM_LENGTH,0)
      ,CL.DATA_TYPE = ISC.DATA_TYPE
   FROM #ColumnList CL
   INNER JOIN INFORMATION_SCHEMA.COLUMNS ISC
     ON CL.COLUMN_NAME = ISC.COLUMN_NAME
  WHERE ISC.TABLE_NAME = @TABLE_NAME
  AND ISC.TABLE_SCHEMA = @TABLE_SCHEMA
 END
-- If test for list of column names
ELSE
 BEGIN
 -- Use all column names, to avoid filtering
  IF @TextCol = 1
   BEGIN
    INSERT INTO #ColumnList (COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH)
     SELECT COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH FROM
INFORMATION_SCHEMA.COLUMNS
     WHERE DATA_TYPE IN ('varchar', 'nvarchar', 'char', 'nchar', 'binary')
     AND TABLE_NAME = @TABLE_NAME
     AND TABLE_SCHEMA = @TABLE_SCHEMA
     AND CHARACTER_MAXIMUM_LENGTH > 0
   END
 IF @NumCol = 1
  BEGIN
   INSERT INTO #ColumnList (COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH)
   SELECT COLUMN_NAME, DATA_TYPE, ISNULL(CHARACTER_MAXIMUM_LENGTH,0) FROM
INFORMATION_SCHEMA.COLUMNS
   WHERE DATA_TYPE IN ('numeric', 'int', 'bigint', 'tinyint', 'smallint',
```

```
'decimal', 'money', 'smallmoney', 'float','real')
   AND TABLE_NAME = @TABLE_NAME
   AND TABLE_SCHEMA = @TABLE_SCHEMA
  END
 IF @DateCol = 1
  BEGIN
   INSERT INTO #ColumnList (COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH)
   SELECT COLUMN_NAME, DATA_TYPE, ISNULL(CHARACTER_MAXIMUM_LENGTH,0) FROM
INFORMATION_SCHEMA.COLUMNS
   WHERE DATA_TYPE IN ('Date', 'DateTime', 'SmallDateTime', #39;DateTime2',
'time')
   AND TABLE_NAME = @TABLE_NAME
   AND TABLE_SCHEMA = @TABLE_SCHEMA
  END
IF @LOBCol = 1
 BEGIN
  INSERT INTO #ColumnList (COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH)
   SELECT COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH FROM
INFORMATION_SCHEMA.COLUMNS
   WHERE DATA_TYPE IN ('varchar', 'nvarchar', 'varbinary', 'xml')
   AND TABLE_NAME = @TABLE_NAME
   AND TABLE_SCHEMA = @TABLE_SCHEMA
   AND CHARACTER_MAXIMUM_LENGTH = -1
 END
END
-- Else test to get all column names
-------------------------------------------------------------------------
-- Test that there are columns to analyse
SELECT @ColumnCheck = COUNT (*) FROM #ColumnList WHERE DATA_TYPE IS NOT NULL
IF @ColumnCheck = 0
 BEGIN
  RAISERROR('The columns do not exist in the selected database or no columns are
selected',16,1)
  RETURN
 END
-------------------------------------------------------------------------
-- Create Temp table used to hold profiling data
-------------------------------------------------------------------------
IF OBJECT_ID('tempdb..#ProfileData') IS NOT NULL
 DROP TABLE tempdb..#ProfileData;
 CREATE TABLE #ProfileData
 (
  TABLE_SCHEMA NVARCHAR(128),
  TABLE_NAME NVARCHAR(128),
  COLUMN_NAME NVARCHAR(128),
  ColumnDataLength INT,
  DataType VARCHAR(128),
  MinDataLength BIGINT,
  MaxDataLength BIGINT,
  AvgDataLength BIGINT,
  MinDate SQL_VARIANT,
  MaxDate SQL_VARIANT,
  NoDistinct BIGINT,
  NoNulls NUMERIC(32,4),
  NoZeroLength NUMERIC(32,4),
  PercentageNulls NUMERIC(9,4),
  PercentageZeroLength NUMERIC(9,4),
  NoDateWithHourminuteSecond BIGINT NULL,
  NoDateWithSecond BIGINT NULL,
```

```
  NoIsNumeric BIGINT NULL,
  NoIsDate BIGINT NULL,
  NoAtLimit BIGINT NULL,
  IsFK BIT NULL DEFAULT 0,
  DataTypeComments NVARCHAR(1500)
 );
-- Get row count
DECLARE @ROWCOUNTTEXT NVARCHAR(1000) = ''
DECLARE @ROWCOUNTPARAM NVARCHAR(50) = ''
SET @ROWCOUNTTEXT = 'SELECT @ROWCOUNTOUT = COUNT (*) FROM ' +
QUOTENAME(@TABLE_SCHEMA) + '.' + QUOTENAME(@TABLE_NAME) + ' WITH (NOLOCK)'
SET @ROWCOUNTPARAM = '@ROWCOUNTOUT INT OUTPUT'
EXECUTE sp_executesql @ROWCOUNTTEXT, @ROWCOUNTPARAM, @ROWCOUNTOUT = @ROWCOUNT
OUTPUT
-------------------------------------------------------------------------
-- Test that there are records to analyse
IF @ROWCOUNT = 0
 BEGIN
  RAISERROR('There is no data in the table to analyse',16,1)
  RETURN
 END
-------------------------------------------------------------------------
-- Define the dynamic SQL used for each column to analyse
-------------------------------------------------------------------------
SET @SQLMETADATA = 'INSERT INTO #ProfileData
(ColumnDataLength,COLUMN_NAME,TABLE_SCHEMA,TABLE_NAME,DataType,MaxDataLength,MinDat

DECLARE SQLMETADATA_CUR CURSOR LOCAL FAST_FORWARD FOR
 SELECT COLUMN_NAME, CHARACTER_MAXIMUM_LENGTH, DATA_TYPE FROM #ColumnList
OPEN SQLMETADATA_CUR
FETCH NEXT FROM SQLMETADATA_CUR INTO @COLUMN_NAME, @CHARACTER_MAXIMUM_LENGTH,
@DATA_TYPE
WHILE @@FETCH_STATUS = 0
 BEGIN
  SET @SQLMETADATA = @SQLMETADATA +'
  SELECT TOP 100 PERCENT ' + CAST(@CHARACTER_MAXIMUM_LENGTH AS VARCHAR(20)) + '
,''' + QUOTENAME(@COLUMN_NAME) + '''
  ,''' + QUOTENAME(@TABLE_SCHEMA) + '''
  ,''' + QUOTENAME(@TABLE_NAME) + '''
  ,''' + @DATA_TYPE + ''''
   + CASE
      WHEN @DATA_TYPE IN ('varchar', 'nvarchar', 'char', 'nchar')
           AND @CHARACTER_MAXIMUM_LENGTH >= 0
             THEN + '
  , MAX(LEN(' + QUOTENAME(@COLUMN_NAME) + '))
  , MIN(LEN(' + QUOTENAME(@COLUMN_NAME) + '))
  , AVG(LEN(' + QUOTENAME(@COLUMN_NAME) + '))
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,(SELECT COUNT (*) from '
   + QUOTENAME(@TABLE_SCHEMA) + '.' + QUOTENAME(@TABLE_NAME) + ' WHERE ISNUMERIC('
+ QUOTENAME(@COLUMN_NAME) + ') = 1)
  ,(SELECT COUNT (*) from ' + QUOTENAME(@TABLE_SCHEMA) + '.' +
QUOTENAME(@TABLE_NAME) + ' WHERE ISDATE(' + QUOTENAME(@COLUMN_NAME) + ') = 1) '
  WHEN @DATA_TYPE IN ('numeric', 'int', 'bigint', 'tinyint', 'smallint',
'decimal', 'money', 'smallmoney', 'float','real') THEN + '
  ,MAX(' + QUOTENAME(@COLUMN_NAME) + ')
```

```
  ,MIN(' + QUOTENAME(@COLUMN_NAME) + ')
  ,AVG(CAST(' + QUOTENAME(@COLUMN_NAME) + ' AS NUMERIC(36,2)))
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,NULL '
   WHEN @DATA_TYPE IN ('DateTime', 'SmallDateTime') THEN + '
  ,NULL
  ,NULL
  ,NULL
  ,MAX(' + QUOTENAME(@COLUMN_NAME) + ')
  ,MIN(' + QUOTENAME(@COLUMN_NAME) + ')
  ,(SELECT COUNT (*) from '
   + QUOTENAME(@TABLE_SCHEMA) + '.' + QUOTENAME(@TABLE_NAME) + ' WHERE
(CONVERT(NUMERIC(20,12), ' + QUOTENAME(@COLUMN_NAME) + ' ) -
FLOOR(CONVERT(NUMERIC(20,12), ' + QUOTENAME(@COLUMN_NAME) + ')) <> 0))
  ,(SELECT COUNT (*) from '
   + QUOTENAME(@TABLE_SCHEMA) + '.' + QUOTENAME(@TABLE_NAME) + ' WHERE
DATEPART(ss,' + QUOTENAME(@COLUMN_NAME) + ') <> 0 OR DATEPART(mcs,' +
QUOTENAME(@COLUMN_NAME) + ') <> 0)
  ,NULL
  ,NULL '
    WHEN @DATA_TYPE IN ('DateTime2') THEN + '
  ,NULL
  ,NULL
  ,NULL
  ,MAX(' + QUOTENAME(@COLUMN_NAME) + ')
  ,MIN(' + QUOTENAME(@COLUMN_NAME) + ')
  ,NULL
  ,NULL
  ,NULL
  ,NULL '
   WHEN @DATA_TYPE IN ('Date') THEN + '
  ,NULL
  ,NULL
  ,NULL
  ,MAX('
   + QUOTENAME(@COLUMN_NAME) + ')
  ,MIN('
  + QUOTENAME(@COLUMN_NAME) + ')
  ,NULL
  ,NLL
  ,NULL
  ,NULL '
   WHEN @DATA_TYPE IN ('xml') THEN + '
  ,MAX(LEN(CAST(' + QUOTENAME(@COLUMN_NAME) + ' AS NVARCHAR(MAX))))
  ,MIN(LEN(CAST(' + QUOTENAME(@COLUMN_NAME) + ' AS NVARCHAR(MAX))))
  ,AVG(LEN(CAST(' + QUOTENAME(@COLUMN_NAME) + ' AS NVARCHAR(MAX))))
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,NULL '
  WHEN @DATA_TYPE IN ('varbinary','varchar','nvarchar') AND
@CHARACTER_MAXIMUM_LENGTH = -1 THEN + '
  ,MAX(LEN(' + QUOTENAME(@COLUMN_NAME) + '))
```

```
  ,MIN(LEN(' + QUOTENAME(@COLUMN_NAME) + '))
  ,AVG(LEN(' + QUOTENAME(@COLUMN_NAME) + '))
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,NULL '
   WHEN @DATA_TYPE IN ('binary') THEN + '
  ,MAX(LEN(' + QUOTENAME(@COLUMN_NAME) + '))
  ,MIN(LEN(' + QUOTENAME(@COLUMN_NAME) + '))
  ,AVG(LEN(' + QUOTENAME(@COLUMN_NAME) + '))
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,NULL '
   WHEN @DATA_TYPE IN ('time') THEN + '
  ,NULL
  ,NULL
  ,NULL
  ,MAX(' + QUOTENAME(@COLUMN_NAME) + ')
  ,MIN(' + QUOTENAME(@COLUMN_NAME) + ')
  ,NULL
  ,NULL
  ,NULL
  ,NULL '
   ELSE + '
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,NULL
  ,NULL '
  END + '
  ,(SELECT COUNT(*) FROM ' + QUOTENAME(@TABLE_SCHEMA) + '.' +
QUOTENAME(@TABLE_NAME) + ' WHERE ' + QUOTENAME(@COLUMN_NAME) + ' IS NULL)'
   + CASE
   WHEN @DATA_TYPE IN ('varchar', 'nvarchar', 'char', 'nchar') THEN + '
  ,(SELECT COUNT(*) FROM ' + QUOTENAME(@TABLE_SCHEMA) + '.' +
QUOTENAME(@TABLE_NAME) +  ' WHERE LEN(LTRIM(RTRIM(' + QUOTENAME(@COLUMN_NAME) +
'))) = '''')'
   ELSE + '
  ,NULL'
   END + '
  ,(SELECT COUNT(DISTINCT ' + QUOTENAME(@COLUMN_NAME) + ') FROM ' +
QUOTENAME(@TABLE_SCHEMA) + '.' + QUOTENAME(@TABLE_NAME) + ' WHERE ' +
QUOTENAME(@COLUMN_NAME) + ' IS NOT NULL )
  FROM ' + QUOTENAME(@TABLE_SCHEMA) + '.' + QUOTENAME(@TABLE_NAME) + ' WITH
(NOLOCK)
  UNION'
 FETCH NEXT FROM SQLMETADATA_CUR INTO @COLUMN_NAME, @CHARACTER_MAXIMUM_LENGTH,
@DATA_TYPE
END
CLOSE SQLMETADATA_CUR
```

```
DEALLOCATE SQLMETADATA_CUR
SET @SQLMETADATA = LEFT(@SQLMETADATA, LEN(@SQLMETADATA) -5)
EXEC (@SQLMETADATA)
------------------------------------------------------------------------
-- Final Calculations
------------------------------------------------------------------------
-- Indicate Foreign Keys
; WITH FK_CTE (FKColumnName)
AS
(
 SELECT
   DISTINCT CU.COLUMN_NAME
  FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS TC
   INNER JOIN INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE CU
     ON TC.CONSTRAINT_NAME = CU.CONSTRAINT_NAME
     AND TC.TABLE_SCHEMA = CU.TABLE_SCHEMA
     AND TC.TABLE_NAME = CU.TABLE_NAME
     AND TC.TABLE_SCHEMA = @TABLE_SCHEMA
     AND TC.TABLE_NAME = @TABLE_NAME
     AND CONSTRAINT_TYPE = 'FOREIGN KEY'
)
UPDATE P
 SET P.IsFK = 1
 FROM #ProfileData P
  INNER JOIN FK_CTE CTE
   ON P.COLUMN_NAME = CTE.FKColumnName
-- Calculate percentages
UPDATE #ProfileData
 SET PercentageNulls = (NoNulls / @ROWCOUNT) * 100
    ,PercentageZeroLength = (NoZeroLength / @ROWCOUNT) * 100
-- Add any comments
-- Datatype suggestions
-- First get number of records where a variation could be an anomaly
SET @DataTypeVariance = ROUND((@ROWCOUNT * @DataTypePercentage) / 100, 0)
UPDATE #ProfileData
  SET DataTypeComments = 'Possibly could be one of the DATE types. '
 WHERE NoIsDate BETWEEN (@ROWCOUNT -@DataTypeVariance) AND (@ROWCOUNT +
@DataTypeVariance)
 AND DataType IN ('varchar', 'nvarchar', 'char', 'nchar')
UPDATE #ProfileData
  SET DataTypeComments = 'Possibly could be one of the NUMERIC types. '
 WHERE NoIsNumeric BETWEEN (@ROWCOUNT -@DataTypeVariance) AND (@ROWCOUNT +
@DataTypeVariance)
 AND DataType IN ('varchar', 'nvarchar', 'char', 'nchar')
UPDATE #ProfileData
  SET DataTypeComments = 'Possibly could be INT type. '
 WHERE MinDataLength >= -2147483648
 AND MaxDataLength <= 2147483648
 AND DataType IN ('bigint')

UPDATE #ProfileData
  SET DataTypeComments = 'Possibly could be SMALLINT type. '
 WHERE MinDataLength >= -32768
 AND MaxDataLength <= 32767
 AND DataType IN ('bigint','int')
UPDATE #ProfileData
  SET DataTypeComments = 'Possibly could be TINYINT type. '
 WHERE MinDataLength >= 0
 AND MaxDataLength <= 255
```

```
 AND DataType IN ('bigint','int','smallint')
UPDATE #ProfileData
  SET DataTypeComments = 'Possibly could be SMALLDATE type. '
 WHERE NoDateWithSecond = 0
 AND MinDate >= '19000101'
 AND MaxDate <= '20790606'
 AND DataType IN ('datetime','datetime2')
UPDATE #ProfileData
  SET DataTypeComments = 'Possibly could be DATE type (SQL Server 2008 only). '
 WHERE NoDateWithHourminuteSecond = 0
 AND DataType IN ('datetime','datetime2')
UPDATE #ProfileData
  SET DataTypeComments = 'Possibly could be DATETIME type. '
 WHERE MinDate >= '17530101'
 AND MaxDate <= '99991231'
 AND DataType IN ('datetime2')
-- Empty column suggestions
UPDATE #ProfileData
  SET DataTypeComments = ISNULL(DataTypeComments,'') + 'Seems empty - is it
required? '
 WHERE (PercentageNulls = 100 OR PercentageZeroLength = 100)
 AND IsFK = 0
-- Null column suggestions
UPDATE #ProfileData
  SET DataTypeComments = ISNULL(DataTypeComments,'') + 'There is a large
percentage of NULLs - attention may be required. '
 WHERE PercentageNulls >= @NullBoundaryPercent
-- Distinct value suggestions
UPDATE #ProfileData
  SET DataTypeComments = ISNULL(DataTypeComments,'') + 'Few distinct elements -
potential for reference/lookup table (contains NULLs).'
 WHERE NoDistinct < @DistinctValuesMinimum
 AND @ROWCOUNT > @DistinctValuesMinimum
 AND IsFK = 0
 AND PercentageNulls <> 100
 AND NoNulls <> 0
-- FK suggestions
UPDATE #ProfileData
  SET DataTypeComments = ISNULL(DataTypeComments,'') + 'Few distinct elements -
potential for Foreign Key.'
 WHERE NoDistinct < @DistinctValuesMinimum
 AND @ROWCOUNT > @DistinctValuesMinimum
 AND IsFK = 0
 AND NoNulls = 0
 AND DataType NOT LIKE '%Date%'
 AND DataType <> 'Time'
-- Filestream suggestions
UPDATE #ProfileData
  SET DataTypeComments = 'Possibly a good candidate for FILESTREAM (SQL Server
2008 only).'
 WHERE AvgDataLength >= 1000000
 AND DataType IN ('varbinary')
 AND ColumnDataLength = -1
UPDATE #ProfileData
  SET DataTypeComments = 'Possibly not a good candidate for FILESTREAM (SQL Server
2008 only).'
 WHERE AvgDataLength < 1000000
 AND DataType IN ('varbinary')
 AND ColumnDataLength = -1
```

```
-- Sparse Column Suggestions
IF OBJECT_ID('tempdb..#SparseThresholds') IS NOT NULL
  DROP TABLE tempdb..#SparseThresholds;
  CREATE TABLE #SparseThresholds (DataType VARCHAR(128), Threshold NUMERIC(9,4))
  INSERT INTO #SparseThresholds (DataType, Threshold)
   VALUES
    ('tinyint',86),
    ('smallint',76),
    ('int',64),
    ('bigint',52),
    ('real',64),
    ('float',52),
    ('money',64),
    ('smallmoney',64),
    ('smalldatetime',52),
    ('datetime',52),
    ('uniqueidentifier',43),
    ('date',69),
    ('datetime2',52),
    ('decimal',42),
    ('nuumeric',42),
    ('char',60),
    ('varchar',60),
    ('nchar',60),
    ('nvarchar',60),
    ('binary',60),
    ('varbinary',60),
    ('xml',60)
; WITH Sparse_CTE (COLUMN_NAME, SparseComment)
AS
(
SELECT
  P.COLUMN_NAME
 ,CASE
  WHEN P.PercentageNulls >= T.Threshold THEN 'Could benefit from sparse columns. '
  ELSE ''
  END AS SparseComment
FROM #ProfileData P
 INNER JOIN #SparseThresholds T
  ON P.DataType = T.DataType
)
UPDATE PT
  SET PT.DataTypeComments =
      CASE WHEN PT.DataTypeComments IS NULL THEN CTE.SparseComment
           ELSE ISNULL(PT.DataTypeComments,'') + CTE.SparseComment + '. '
      END
 FROM #ProfileData PT
  INNER JOIN Sparse_CTE CTE
   ON PT.COLUMN_NAME = CTE.COLUMN_NAME
-------------------------------------------------------------------------------
-- Optional advanced analysis
-------------------------------------------------------------------------------
IF @AdvancedAnalysis = 1
 BEGIN
-------------------------------------------------------------------------------
-- Data at data boundaries
-------------------------------------------------------------------------------
  IF OBJECT_ID('tempdb..#LimitTest') IS NOT NULL
    DROP TABLE tempdb..#LimitTest;
```

```
    CREATE TABLE #LimitTest (COLUMN_NAME VARCHAR(128), NoAtLimit BIGINT);
    DECLARE @advancedtestSQL VARCHAR(MAX) = 'INSERT INTO #LimitTest (COLUMN_NAME,
NoAtLimit)' + CHAR(13)
    SELECT @advancedtestSQL = @advancedtestSQL + 'SELECT '''+ COLUMN_NAME + ''',
COUNT('+ COLUMN_NAME + ') FROM ' + @TABLE_SCHEMA + '.' + @TABLE_NAME +
     CASE
       WHEN DataType IN ('numeric', 'int', 'bigint', 'tinyint', 'smallint',
'decimal', 'money', 'smallmoney', 'float','real') THEN ' WHERE '+ COLUMN_NAME + '
= ' + CAST(ISNULL(MaxDataLength,0) AS VARCHAR(40)) + ' OR '+ COLUMN_NAME + ' = ' +
CAST(ISNULL(MinDataLength,0) AS VARCHAR(40)) + CHAR(13) + ' UNION' + CHAR(13)
       ELSE ' WHERE LEN('+ COLUMN_NAME + ') = ' + CAST(ISNULL(MaxDataLength,0) AS
VARCHAR(40)) + ' OR LEN('+ COLUMN_NAME + ') = ' + CAST(ISNULL(MinDataLength,0) AS
VARCHAR(40)) + CHAR(13) + ' UNION' + CHAR(13)
     END
    FROM #ProfileData
    WHERE DataType IN ('numeric', 'int', 'bigint', 'tinyint', 'smallint',
'decimal', 'money', 'smallmoney', 'float','real','varchar', 'nvarchar', 'char',
'nchar', 'binary')
    SET @advancedtestSQL = LEFT(@advancedtestSQL,LEN(@advancedtestSQL) -6)
    EXEC (@advancedtestSQL)
    UPDATE M
      SET M.NoAtLimit = T.NoAtLimit
         ,M.DataTypeComments =
           CASE
             WHEN CAST(T.NoAtLimit AS NUMERIC(36,2)) / CAST(@ROWCOUNT AS
NUMERIC(36,2)) >= @BoundaryPercent THEN ISNULL(M.DataTypeComments,'') + 'Large
numbers of data elements at the max/minvalues. '
             ELSE M.DataTypeComments
           END
    FROM #ProfileData M
     INNER JOIN #LimitTest T
      ON M.COLUMN_NAME = T.COLUMN_NAME
   ------------------------------------------------------------------------
   -- Domain analysis
   ------------------------------------------------------------------------
   IF OBJECT_ID('tempdb..#DomainAnalysis') IS NOT NULL
     DROP TABLE tempdb..#DomainAnalysis;
   CREATE TABLE #DomainAnalysis
   (
    DomainName NVARCHAR(128)
   ,DomainElement NVARCHAR(4000)
   ,DomainCounter BIGINT
   ,DomainPercent NUMERIC(7,4)
   );
   DECLARE @DOMAINSQL VARCHAR(MAX) = 'INSERT INTO #DomainAnalysis (DomainName,
DomainElement, DomainCounter) '
   DECLARE SQLDOMAIN_CUR CURSOR LOCAL FAST_FORWARD FOR
     SELECT COLUMN_NAME, DataType
          FROM #ProfileData
           WHERE NoDistinct < @DistinctValuesMinimum
   OPEN SQLDOMAIN_CUR
   FETCH NEXT FROM SQLDOMAIN_CUR INTO @COLUMN_NAME, @DATA_TYPE
   WHILE @@FETCH_STATUS = 0
    BEGIN
     SET @DOMAINSQL = @DOMAINSQL + 'SELECT ''' + @COLUMN_NAME + ''' AS DomainName,
CAST( '+ @COLUMN_NAME + ' AS VARCHAR(4000)) AS DomainElement, COUNT(ISNULL(CAST('
+ @COLUMN_NAME + ' AS NVARCHAR(MAX)),'''')) AS DomainCounter FROM ' +
@TABLE_SCHEMA + '.' + @TABLE_NAME + ' GROUP BY ' + @COLUMN_NAME + ''
     + ' UNION '
```

```
     FETCH NEXT FROM SQLDOMAIN_CUR INTO @COLUMN_NAME, @DATA_TYPE
   END
  CLOSE SQLDOMAIN_CUR
  DEALLOCATE SQLDOMAIN_CUR
  SET @DOMAINSQL = LEFT(@DOMAINSQL, LEN(@DOMAINSQL) -5) + ' ORDER BY DomainName
ASC, DomainCounter DESC '
   EXEC (@DOMAINSQL)
   -- Now calculate percentages (this appraoch is faster than doing it when
performing the domain analysis)
   ; WITH DomainCounter_CTE (DomainName, DomainCounterTotal)
   AS
  (
   SELECT DomainName, SUM(ISNULL(DomainCounter,0)) AS DomainCounterTotal
    FROM #DomainAnalysis
    GROUP BY DomainName
  )
  UPDATE D
    SET D.DomainPercent = (CAST(D.DomainCounter AS NUMERIC(36,4)) /
CAST(CTE.DomainCounterTotal AS NUMERIC(36,4))) * 100
   FROM #DomainAnalysis D
    INNER JOIN DomainCounter_CTE CTE
     ON D.DomainName = CTE.DomainName
   WHERE D.DomainCounter <> 0
 END
-- Advanced analysis
-------------------------------------------------------------------------
-- Output results from the profile and domain data tables
-------------------------------------------------------------------------
select
   *
 from #ProfileData
IF @AdvancedAnalysis = 1
 BEGIN
  select
    *
   from #DomainAnalysis
 END
END TRY
BEGIN CATCH
 SELECT
  ERROR_NUMBER() AS ErrorNumber
 ,ERROR_SEVERITY() AS ErrorSeverity
 ,ERROR_STATE() AS ErrorState
 ,ERROR_PROCEDURE() AS ErrorProcedure
 ,ERROR_LINE() AS ErrorLine
 ,ERROR_MESSAGE() AS ErrorMessage;

END CATCH
```

This script will return the following columns:

- TABLE_SCHEMA
- TABLE_NAME
- COLUMN_NAME
- ColumnDataLength
- DataType
- MinDataLength

- MaxDataLength
- AvgDataLength
- MinDate
- MaxDate
- NoDistinct
- NoNulls
- NoZeroLength
- PercentageNulls
- PercentageZeroLength
- NoDateWithHourminuteSecond
- NoDateWithSecond
- NoIsNumeric
- NoIsDate
- NoAtLimit
- IsFK
- DataTypeComments

If you profile the data distribution, the results will include the following columns:

- DomainName
- DomainElement
- DomainCounter
- DomainPercent

The results run from the CarSales table (given in the script "CarSales.Sql") are shown in the following screen capture – which is split to give the results in a more readable format:

Results | Messages

	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ColumnDataLength	DataType	MinDataLength	MaxDataLength	AvgDataLength	MinDate	MaxDate	NoDistinct	NoNulls	NoZeroLength
1	[dbo]	[client]	[ClientSince]	0	smalldatetime	NULL	NULL	NULL	1961-01-21 00:00:00.000	2011-04-01 00:00:00.000	5	0.0000	NULL
2	[dbo]	[client]	[Country]	0	tinyint	1	3	2	NULL	NULL	3	0.0000	NULL
3	[dbo]	[client]	[ID]	0	int	3	7	5	NULL	NULL	5	0.0000	NULL
4	[dbo]	[client]	[ClientSize]	10	varchar	1	1	1	NULL	NULL	3	0.0000	0.0000
5	[dbo]	[client]	[PostCode]	10	varchar	5	8	6	NULL	NULL	4	1.0000	0.0000
6	[dbo]	[client]	[ClientType]	20	varchar	7	8	7	NULL	NULL	2	0.0000	0.0000
7	[dbo]	[client]	[Address1]	50	varchar	8	17	12	NULL	NULL	4	1.0000	0.0000
8	[dbo]	[client]	[Address2]	50	varchar	17	17	17	NULL	NULL	1	4.0000	0.0000
9	[dbo]	[client]	[County]	50	varchar	4	8	6	NULL	NULL	3	1.0000	0.0000
10	[dbo]	[client]	[Town]	50	varchar	5	9	7	NULL	NULL	5	0.0000	0.0000
11	[dbo]	[client]	[ClientName]	150	nvarchar	8	14	10	NULL	NULL	5	0.0000	0.0000

PercentageNulls	PercentageZeroLength	NoDateWithHourminuteSecond	NoDateWithSecond	NoIsNumeric	NoIsDate	NoAtLimit	IsFK	DataTypeComments
0.0000	NULL	0	0	NULL	NULL	NULL	0	
0.0000	NULL	NULL	NULL	NULL	NULL	3	0	Large numbers of data elements at the max/minvalues.
0.0000	NULL	NULL	NULL	NULL	NULL	2	0	Possibly could be TINYINT type. .
0.0000	0.0000	NULL	NULL	0	0	5	0	Large numbers of data elements at the max/minvalues.
20.0000	0.0000	NULL	NULL	1	0	2	0	
0.0000	0.0000	NULL	NULL	0	0	5	0	Large numbers of data elements at the max/minvalues.
20.0000	0.0000	NULL	NULL	0	0	2	0	
80.0000	0.0000	NULL	NULL	0	0	1	0	Could benefit from sparse columns.
20.0000	0.0000	NULL	NULL	0	0	2	0	
0.0000	0.0000	NULL	NULL	0	0	2	0	
0.0000	0.0000	NULL	NULL	0	0	2	0	

Further Ideas

As I have said, this script is one that I use fairly frequently, but it is only a basis for analysis and should only be used as a starting point for data profiling and not a magic bullet. You can extend the data type suggestions to suit your requirements. Remember

that text fields can contain duff data, which no amount of automated profiling can pick up.

It can also be useful to look at the data distribution for numeric data, assuming that any edge cases are errors, and need not be stored as part of the data. There could be a tiny number of elements at the limits for the data type, so the field could possibly contain inappropriate data and the type be altered to a smaller data type. You can direct the output from this script to a permanent table, if you prefer. Indeed this script can be wrapped in an outer loop to cycle through a list of tables. I will leave you to do that, however, if you need to.

There are, of course, many other approaches to data profiling. You have the SSIS Profiling task, for instance which is extremely useful. There are also third party tools which will provide extensive profiling of your source data.

The test table which I have used here is taken from the sample database in my book “SQL Server 2012 Data Integration Recipes”. It is used with permission of Apress.

Finally, my thanks to my old colleague Steven Wilbur for helping me with suggestions for this script.