Optimizing an Overly Complex SQL Query

A Data and Analytic Solution (DAS) client routinely ran an essential production job that started with a query to retrieve data from an SQL Server®[1] database.[2] That query was explicitly[3] passed from SAS®[4] to the RDBMS[5] and ran in 3 hours and 24 minutes[6] to produce a SAS® dataset named **Country**. It joined 20 tables that ranged in size from three rows to nine million rows. An outer series of joins combined data from 11 tables and a nested join combined data from the other nine tables. **The optimization described in this paper of the complex SQL query reduced the run time to less than eight minutes – a 95% efficiency gain.**

The SQL language provides a high-level declarative interface, so the user only specifies what the result is to be, leaving the actual optimization and decisions on how to execute the query to the RDBMS.[7] The query optimizer attempts to determine the most efficient way to execute a query. It considers possible query plans for a given query and attempts to determine which of those plans will be the most efficient. This set of query plans is formed by examining the possible access paths[8] and join[9] techniques.

Transact-SQL[10] is central to using Microsoft™ SQL Server®. Its query optimizer will develop at most 256 plans and picks its estimation of the best plan.[11] Unfortunately, the optimizer was not able to develop a plan for this join of 20 tables that was efficient.

The following is a detailed explanation of how DAS optimized the SQL query and reduced processing time by 95 percent. The information is both comprehensive and detailed with sufficient code examples to make the task reproducible. For further information and comments, please contact Gary McQuown at mcquown@DASconsultants.com . Additional papers may be found on our website at www.DASconsultants.com.

[1] SQL Server is a registered trademark of Microsoft Corporation.
[2] See Appendix 1.
[3] Explicit SQL pass-through passes database-specific SQL untouched to the database.
[4] SAS is a registered trademark of SAS Institute, Inc. in the USA and other countries.
[5] A Relational Database Management System (RDBMS) is a software application to manage a collection of relations. Informally, each relation represents a table of values – each row represents a real-world entity and each column represents a common attribute for each of those entities. The relational model was introduced by Ted Codd of IBM Research in 1970.
[6] The program used 3 hours and 59 minutes of CPU time on the multiple-CPU Windows Server.
[7] Elmasri, Ramez and Shamkant B. Navathe. Fundamentals of Database Systems, 3rd Edition. Addison-Wesley, 2000. Page 244.
[8] Access paths include primary index access, secondary index access, and full file scan.
[9] Relational table join techniques include merge join, hash join, and product join.
[10] Transact-SQL (T-SQL) is Microsoft's and Sybase's proprietary implementation of SQL.
[11] By comparison, DB2's optimizer will develop and evaluate up to 32,768 plans.

Initial Evaluation

Without access to the RDBMS where the data is stored, the off-site DAS analyst provided SAS® scripts to an on-site DAS employee who returned the output for evaluation. Initially, we learned about the size of the tables and the columns in those tables.

The outer series of joins combined the following 11 tables:

- **AdHocQuestionnaireDefs** had about 300 rows. Only rows where
  - the ***RootAdHocQuestionnaireDefId*** key was **10709** and
  - the ***IsActive*** flag was set to **1**

  were used.
- **RefAdHocQuestionnaireDefStatuses** had 3 rows.
- **AdHocQuestionGroupDefs** had about 350 rows.
- **RefAdHocQuestionnaireTypes** had about 15 rows.
- **AdHocQuestionDefs** had about 2,600 rows.
- **RefAdHocQuestionTypes** had 10 rows.
- **AdHocQuestionnaires** had about 250,000 rows.
- **AdHocQuestionnaireIdentifiers** had about 250,000 rows.
- **AdHocQuestions** had about 3,500,000 rows.
- **AdHocAnswers** had about 700,000 rows.
- **AdHocAnswerDefs** had about 4,000 rows.

The output from this outer series of joins consisted of about 1,400,000 rows and 12 columns. If the optimizer can come up with a fairly efficient plan for the join, it should run in just a few minutes.

The nested join combined the following nine tables:

- **InspectionTasks** had about 130 rows. Only rows where the ***InspectionTaskCode*** was either **06A03** or **06A04** were used.
- **EstablishmentTaskLists** had about 5,000,000 rows.
- **EstablishmentNumberOrgLevel** had about 8,000 rows.
- **EstablishmentShiftXInspectTask** had about 9,000,000 rows.
- **EstablishmentShifts** had about 9,000 rows.
- **RefInspectionTaskJustifications** had about 30 rows.
- **InspectionResults** had about 8,000,000 rows.
- **Users** had about 12,000 rows.
- **Areas** had about 60 rows. Only rows where the ***BusinessUnitId*** was set to **1** were used.

The output from this nested series of joins consisted of about 18,000 rows and 12 columns. If the optimizer can come up with a fairly efficient plan for the join, it should run in less than a minute.

Both of these queries return character variables from SQL Server® *varChar*[12] columns that are extremely long given the lengths of the character strings in these columns. These long character variables should be shortened after the download.

After that evaluation, the DAS analyst estimated that a more-efficient process should be able to create the Country dataset in about 15 minutes.

Creating a Table of Questionnaire, Question, and Answer Data

First, we queried the RDBMS to create a SAS® dataset named **Questionnaire**. This dataset consisted of about 1.4 million rows with a foreign key named ***TaskConnector*** that would be used to link to the data from the nested query. SAS® implicitly passed the query to the RDBMS because all of the code in the SAS® SQL query could be converted to Transact-SQL. The query ran in about four minutes.

Long Character Variables

TaskConnector exists in the RDBMS as a 4000-byte *varChar* variable. However, SAS® has no *varChar* data type and the key arrives in SAS® as a 4,000 byte character string. Examination of the ***TaskConnector*** keys revealed that each consisted of either

- a nine-digit decimal integer (in about 3% of the rows) or
- a decimal integer of one to three digits followed immediately by the canonical representation of a UUID[13] – with either upper-case letters or lower-case letters.

The UUID part of the two-part ***TaskConnector*** keys sometimes used upper-case letters to represent the six largest hexadecimal digits and sometimes it used lower-case letters. While lower-case letters seem to be the standard across most applications, SQL Server® uses an upper-case standard.[14]

A SAS® SQL query implicitly[15] passed to the SQL Server® dataset to join the 11 tables in the outer join produced the **QandA** table in about 3½ minutes.[16]

[12] In SQL Server, the *varChar* data type is a variable-length, non-Unicode string data with a maximum length that can be specified from 1 through 8,000. The storage size is the actual length of the data entered + 2 bytes.
[13] A universally unique identifier (UUID) is a 128-bit integer. That is, it is an integer with 2^{128} possible unique values. The intent is to enable distributed systems to uniquely identify information without significant central coordination. In its canonical form, a UUID is represented by 32 hexadecimal digits, displayed in five groups separated by hyphens, in the form 8-4-4-4-12 for a total of 36 characters (32 alphanumeric characters and four hyphens).
[14] SAS uses the lower-case standard for its canonical form of the UUID.
[15] With implicit SQL pass-through (identified by the use of a LIBNAME statement pointing to the relational database) SAS will attempt to convert the SAS SQL scripts to SQL that the target database can understand.
[16] See Appendix 2.

The SQL Server® database uses the *varChar* data type to allow short character strings to be stored efficiently in columns defined to allow long strings. However, SAS® does not have a *varChar* equivalent and those columns come to SAS® as fixed-length variables padded with blanks to the full length of the defined *varChar* column in SQL Server®. Thus, character strings in the **QandA** table were significantly longer than needed. Specifically

- ***Name*** was 50 bytes but the longest string was 29 bytes,
- ***Title*** was 100 bytes but the longest string was 19 bytes,
- ***QuestionText*** was 4,000 bytes but the longest string was 207 bytes,
- ***QuestionType*** was 20 bytes but the longest string was 15 bytes,
- ***Answer*** was 4,000 bytes but the longest string was 388 bytes, and
- ***TaskConnector*** was 4,000 bytes but the longest string was 39 characters

We can use the SAS® SQL procedure to create a series of macro variables – each with the length of the longest string found in each of these character variables. Then we can use those macro variables in a SAS® DATA step to shorten the lengths of those character variables. While we have our data in the DATA step, it would be useful to create a variable for the integer portion of the ***TaskConnector*** and another variable for the UUID portion of the ***TaskConnector***.[17]

The query that creates the six macro variables ran in about 23 seconds and the DATA step that shortened the variables ran in about 24 seconds.

Creating a Table of Inspection and Establishment Data

Let's create a dataset of inspection and establishment data – call it **Inspection** – by submitting the nested part of the original query as an independent query. We want to join

- **InspectionTasks**,
- **EstablishmentTaskLists**,
- **EstablishmentNumberOrgLevel**,
- **EstablishmentShiftXInspectTask**,
- **EstablishmentShifts**,
- **RefInspectionTaskJustifications**,
- **InspectionResults**,
- **Users**, and
- **Areas**.

[17] See Appendix 3.

SAS® will implicitly pass a query to the RDBMS if it can be converted to the query language that is used by that RDBMS. However, the **InspectionResults** table in the database has a column named ***EtsbablishmentShiftXInspectTaskId***[18] and that name is 33 characters long while SAS® only allows column names up to 32 characters. So, we must explicitly pass a Transact-SQL query to SQL Server®.

This query ran in about three seconds. It returned about 18 thousand rows and 12 columns.

Long Character Variables

PrimaryEstablishmentNumber came from SQL Server® as a 500-byte character variable but the longest string returned was seven bytes. Other *varChar* values arrived longer then prudent for SAS®. We shortened them to lengths suggested by the data.

Creating the Country Dataset

Finally, we need to join the SAS® **Questionnaire** dataset with the SAS® **Inspection** dataset by matching the ***TaskConnector*** from **Questionnaire** with the ***IdentifierValue*** from **Inspection**. We will sort the output **Country** dataset on the ***GrpSortOrd*** values and then on the ***QDefSortOrd*** values within each level of ***GrpSortOrd***.[19] This query ran in about 20 seconds to produce 1.4 million rows and 20 columns.

Upper-Case and Lower-Case Canonical Representation of UUIDs

Almost 800 thousand of the UUID values in the ***TaskConnector*** column had lower-case letters for the larger hexadecimal digits. These represented about 10 thousand distinct values. In all, we had about 18 thousand distinct case-insensitive values in the UUID section of the ***TaskConnector*** column.

We have about four hundred distinct lower-case ***TaskConnector*** values that match to ***IdentifierValue*** using a case-insensitive join.

The original query concatenates the ***Id*** columns from the **InspectionTasks** and **EstablishmentShiftXInspectTask** tables in a nested join using the following code.

```
select rtrim(it.Id) + rtrim(esit.Id) as IdentifierValue
```

Then the job joins that nested join (alias: **a**) with ***IdentifierValue*** values from the **AdHocQuestionnaireIdentifiers** table.

The ***Id*** column in the **InspectionTasks** table is an integer and the ***Id*** column in the **EstablishmentShiftXInspectTask** table is a *uniqueIdentifier*[20] (UUID) – a 16-byte integer.

[18] Note: The name of the variable is probably a typo and should have been *EstablishmentShiftXInspectTaskId* which is 32 characters long.
[19] See Appendix 7.
[20] SQL Server's uniqueIdentifier data type is a UUID.

The rTrim()[21] function converted the keys to character strings. No problem with the ***Id*** from the **InspectionTasks** table because the result is all decimal digits. However, when rTrim() cast the *uniqueIdentifier* value to a character string, that string had upper-case letters representing the larger hexadecimal digits (e.g.: **B9CA9D3E-7EFC-E111-93FF-005056945884**).[22]

The problem is that some of the ***IdentifierValue*** values in **AdHocQuestionnaireIdentifiers** table have upper-case letters and some have lower-case letters! When used as a key for the equi-join of the tables, case matters!

```
on a.IdentifierValue = AHQI.IdentifierValue
```

Perhaps we should add the following line to the DATA step that shortens the character variables in Questionnaire.[23]

```
TaskConnector = upCase(TaskConnector) ;
```

Conclusion

The original query to download the Country data ran on the 10th of December in about 3½ hours and downloaded 1,178,499 rows. While this was only a single step, the query was simply too complex for the SQL Server® query optimizer.

While our solution increases this single step to seven steps, the total run time is only about 5½ minutes to return 1,450,155 rows.[24] That's about 50 times faster!

[21] The Transact-SQL rTrim() function removes trailing spaces from a character argument.

[22] SAS has a function called uuidGen() which generates a UUID and returns it as either a string of hexadecimal digits in groups separated by hyphens (e.g.: b9ca9d3e-7efc-e111-93ff-005056945884) or as a 16-byte binary string. Notice that the SAS uses lower-case letters to represent the larger hexadecimal digits!

[23] The SAS upCase() function converts all lower-case ASCII letters (including those with diacritical marks) to upper-case.

[24] The number of returned rows increased from the initial run in December because rows were added to the tables that the task queried.

Appendix 1: Original Query

```
Create table work.country as
    select *
    from connection to ph (
        select
            AHQRD.Id as QuestionnaireDefID
          , AHQRD.Name
          , AHQGD.Title
          , AHQGD.SortOrder as GrpSortOrd
          , AHQD.QuestionText
          , AHQD.SortOrder QDefSortOrd
          , RAHQT.Code as QuestionType
          , case when AHA.AnswerText is not null then AHA.AnswerText else AHAD.Label end as Answer
          , AHQD.Id as QuestionDefID
          , AHQR.Id as QuestionnaireID
          , AHQI.IdentifierValue as TaskConnector
          , AHQ.Id as QuestionID
          , a.PrimaryEstablishmentNumber
          , a.EstablishmentName
          , a.DistrictNumber
          , a.Description
          , a.startdate
          , a.inspector
          , a.District
          , a.inspectionresultnumber
        from
            AdHocQuestionnaireDefs AHQRD
            join RefAdHocQuestionnaireDefStatuses RADHQDS
                on RADHQDS.Id = AHQRD.AdHocQuestionnaireDefStatusId
            join AdHocQuestionGroupDefs AHQGD on AHQGD.AdHocQuestionnaireDefId = AHQRD.Id
            join RefAdHocQuestionnaireTypes RAHQRT on RAHQRT.Id = AHQRD.AdHocQuestionnaireTypeId
            join AdHocQuestionDefs AHQD on AHQD.AdHocQuestionGroupDefId = AHQGD.Id
            join RefAdHocQuestionTypes RAHQT on  RAHQT.Id = AHQD.AdHocQuestionTypeId
            join AdHocQuestionnaires AHQR on AHQR.AdHocQuestionnaireDefId = AHQRD.Id
            join AdHocQuestionnaireIdentifiers AHQI on AHQI.AdHocQuestionnaireId = AHQR.Id
            join AdHocQuestions AHQ
                on AHQ.AdHocQuestionDefId = AHQD.Id and AHQ.AdHocQuestionnaireId = AHQR.Id
            left join AdHocAnswers AHA on AHA.AdHocQuestionId = AHQ.Id
            left join AdHocAnswerDefs AHAD on AHA.AdHocAnswerDefId = AHAD.Id
```

```
            left join (
                select
                    rtrim(it.Id) + rtrim(esit.Id) as IdentifierValue
                  , etl.EstablishmentId
                  , es.ShiftTypeId
                  , eno.PrimaryEstablishmentNumber
                  , eno.EstablishmentName
                  , eno.DistrictNumber
                  , esit.InspectorId
                  , ritj.Description
                  , convert(date,ir.StartDate) as StartDate
                  , u.FirstName + ' ' + u.LastName as Inspector
                  , a.Description as District
                  , ir.inspectionresultnumber
                from
                    InspectionTasks it
                    join EstablishmentTaskLists etl on etl.InspectionTaskId = it.Id
                    join EstablishmentNumberOrgLevel eno on eno.EstablishmentID = etl.EstablishmentId
                    join EstablishmentShiftXInspectTask esit on esit.EstablishmentTaskListId = etl.Id
                    left join EstablishmentShifts es on es.Id = esit.EstablishmentShiftId
                    left join RefInspectionTaskJustifications ritj
                        on ritj.Id = esit.InspectionTaskJustificationId
                    join InspectionResults ir with (nolock)
                        on ir.EtsbablishmentShiftXInspectTaskId = esit.Id
                    join Users u on u.ID = ir.InspectorId
                    join (
                        select *
                        from Areas
                        where businessunitid = 1
                    ) a on a.Number = eno.DistrictNumber
                where it.InspectionTaskCode in ('06A03','06A04')
            ) a on a.IdentifierValue = AHQI.IdentifierValue
        where AHQRD.RootAdHocQuestionnaireDefId = 10709 and AHQRD.IsActive = 1
        order by AHQGD.SortOrder, AHQD.SortOrder
    )
;
```

Appendix 2: Query to Create the Questionnaire Dataset

```
%let qxId = 10709 ;
%let isActive = 1 ;
...
LibName ph
    oledb
    udl_file="C:\Users\Public\Documents\OLEDB\PHISPROD.udl"
    dbmax_text=32767
    dbSasLabel=none
;
Proc sql ;
    Create table Questionnaire as
        select
            AdHocQuestionnaireDefs.Id as QuestionnaireDefID
          , AdHocQuestionnaireDefs.Name
          , AdHocQuestionGroupDefs.Title
          , AdHocQuestionGroupDefs.SortOrder as GrpSortOrd
          , AdHocQuestionDefs.QuestionText
          , AdHocQuestionDefs.SortOrder as QDefSortOrd
          , RefAdHocQuestionTypes.Code as QuestionType
          , case
                when AdHocAnswers.AnswerText is null
                    then AdHocAnswerDefs.Label
                else AdHocAnswers.AnswerText
            end as Answer
          , AdHocQuestionDefs.Id as QuestionDefID
          , AdHocQuestionnaires.Id as QuestionnaireID
          , AdHocQuestionnaireIdentifiers.IdentifierValue
                as TaskConnector
          , AdHocQuestions.Id as QuestionISD
```

```
        from
         /* The RefAdHocQuestionnaireDefStatuses and
            RefAdHocQuestionnaireTypes tables contribute no columns
            to the output dataset. */
            (   select
                    Id
                  , Name
                  , AdHocQuestionnaireDefStatusId
                  , AdHocQuestionnaireTypeId
                from ph.AdHocQuestionnaireDefs
                where
                    ( RootAdHocQuestionnaireDefId eq &qxId )
                  & ( IsActive eq &isActive )
            ) as AdHocQuestionnaireDefs join ph.RefAdHocQuestionnaireDefStatuses on (
                RefAdHocQuestionnaireDefStatuses.Id
             eq AdHocQuestionnaireDefs.AdHocQuestionnaireDefStatusId
            ) join ph.AdHocQuestionGroupDefs on (
                AdHocQuestionGroupDefs.AdHocQuestionnaireDefId
             eq AdHocQuestionnaireDefs.Id
            ) join ph.RefAdHocQuestionnaireTypes on (
                RefAdHocQuestionnaireTypes.Id
             eq AdHocQuestionnaireDefs.AdHocQuestionnaireTypeId
            ) join ph.AdHocQuestionDefs on (
                AdHocQuestionDefs.AdHocQuestionGroupDefId
             eq AdHocQuestionGroupDefs.Id
            ) join ph.RefAdHocQuestionTypes on (
                RefAdHocQuestionTypes.Id
             eq AdHocQuestionDefs.AdHocQuestionTypeId
            ) join ph.AdHocQuestionnaires on (
                AdHocQuestionnaires.AdHocQuestionnaireDefId
             eq AdHocQuestionnaireDefs.Id
            ) join ph.AdHocQuestionnaireIdentifiers on (
                AdHocQuestionnaireIdentifiers.AdHocQuestionnaireId
             eq AdHocQuestionnaires.Id
            ) join ph.AdHocQuestions on (
                ( AdHocQuestions.AdHocQuestionDefId eq AdHocQuestionDefs.Id )
              & ( AdHocQuestions.AdHocQuestionnaireId eq AdHocQuestionnaires.Id )
            ) left join ph.AdHocAnswers on (
                AdHocAnswers.AdHocQuestionId eq AdHocQuestions.Id
            ) left join ph.AdHocAnswerDefs as aDef on (
                AdHocAnswers.AdHocAnswerDefId eq AdHocAnswerDefs.Id
            )
        order by
            AdHocQuestionGroupDefs.SortOrder
          , AdHocQuestionDefs.SortOrder
    ;
Quit ;
LibName ph clear ;
```

Appendix 3: Find the Maximum Lengths of the Character Strings in Questionnaire

The following character variables came to SAS with these lengths:

- *Name* 50
- *Title* 100
- *QuestionText* 4,000
- *QuestionType* 20
- *Answer* 4,000
- *TaskConnector* 4,000

```
Proc sql ;
    Select
            max(length(Name)) label="longest Name"
          , max(length(Title)) label="longest Title"
          , max(length(QuestionText)) label="longest QuestionText"
          , max(length(QuestionType)) label="longest QuestionType"
          , max(length(Answer)) label="longest Answer"
          , max(length(TaskConnector)) label="longest TaskConnector"
        into
            :lenName
          , :lenTitle
          , :lenQuestionText
          , :lenQuestionType
          , :lenAnswer
          , :lenTaskConnector
    from QandA ;
Quit ;
```

longest Name	longest Title	longest QuestionText	longest QuestionType	longest Answer	longest Task Connector
29	19	207	15	388	39

Appendix 4: Shorten the Character Variables in Questionnaire

```
 /* The following step will produce the following warning for each of the character
    variables we are shortening.
        Multiple lengths were specified for the variable ... by input data set(s).
        This may cause truncation of data.
    This is exactly what we want.  SAS provides no mechanism to turn off these
    warnings.  We could avoid them by renaming the variables in the SET statement,
    writing their values to new variables, and dropping the renamed variables.  But,
    that's a lot of code just to avoid the warnings. */
    Data Questionnaire( label="Questionnaires, Questions, and Answers" ) ;
        Length
            QuestionnaireDefID  8
            Name               $&lenName
            Title              $&lenTitle
            GrpSortOrd          8
            QuestionText       $&lenQuestionText
            QDefSortOrd         8
            QuestionType       $&lenQuestionType
            Answer             $&lenAnswer
            QuestionDefID       8
            QuestionnaireID     8
            TaskConnector      $&lenTaskConnector
            QuestionID          8
        ;
    Set QandA ;
     /* The UUID part of the TaskConnector expresses the larger hexadecimal digits
        sometimes as upper-case letters and sometimes as lower-case letters.  We need
        to standardize this before we attempt to merge the QandA table with the
        Inspection table - which will have all the UUID values expressed with upper-
        case letters. */
/*        TaskConnector = upCase(TaskConnector) ;*/
    Run ;
    %symDel
        lenName
        lenTitle
        lenQuestionText
        lenQuestionType
        lenAnswer
        lenTaskConnector
    ;
```

Appendix 5: Explicit Pass-Through Query to Create the Inspection Dataset

```
/* The included file contains a script to create the &ph_connect macro
   variable. */
   %include "E:\SAS JobRunEnv\DatabaseExec\Autoexec_Svr1.sas";
   Proc sql &sqlOptions ;
       &ph_connect ;
       Create table Inspection as select * from connection to ph (
           select
               rTrim(InspectionTasks.Id) + rTrim(
                   EstablishmentShiftXInspectTask.Id
               ) as IdentifierValue
             , EstablishmentTaskLists.EstablishmentId
             , EstablishmentShifts.ShiftTypeId
             , EstablishmentNumberOrgLevel.PrimaryEstablishmentNumber
             , EstablishmentNumberOrgLevel.EstablishmentName
             , EstablishmentNumberOrgLevel.DistrictNumber
             , EstablishmentShiftXInspectTask.InspectorId
             , RefInspectionTaskJustifications.Description
             , convert(date,InspectionResults.StartDate) as StartDate
             , Users.FirstName + ' ' + Users.LastName as Inspector
             , Areas.Description as District
             , InspectionResults.inspectionresultnumber
           from
               InspectionTasks join EstablishmentTaskLists on (
                   EstablishmentTaskLists.InspectionTaskId
                 = InspectionTasks.Id
               ) join EstablishmentNumberOrgLevel on (
                   EstablishmentNumberOrgLevel.EstablishmentID
                 = EstablishmentTaskLists.EstablishmentId
               ) join EstablishmentShiftXInspectTask on (
                   EstablishmentShiftXInspectTask.EstablishmentTaskListId
                 = EstablishmentTaskLists.Id
               ) left join EstablishmentShifts on (
                   EstablishmentShifts.Id
                 = EstablishmentShiftXInspectTask.EstablishmentShiftId
               ) left join RefInspectionTaskJustifications on (
                   RefInspectionTaskJustifications.Id
                 = EstablishmentShiftXInspectTask.InspectionTaskJustificationId
               ) join InspectionResults with (nolock) on (
                   InspectionResults.EtsbablishmentShiftXInspectTaskId
                 = EstablishmentShiftXInspectTask.Id
               ) join Users on (
                   Users.ID = InspectionResults.InspectorId
               ) join (
                   select Description , Number
                   from Areas
                   where BusinessUnitId = 1
               ) as Areas on (
                   Areas.Number
                 = EstablishmentNumberOrgLevel.DistrictNumber
               )
           where InspectionTasks.InspectionTaskCode in ('06A03','06A04')
       ) ;
       Disconnect from ph ;
   Quit;
```

Appendix 6: Find the Maximum Lengths of the Character Strings in Inspection

The following character variables came to SAS with these lengths:

- *IdentifierValue* 52
- *PrimaryEstablishmentNumber* 500
- *EstablishmentName* 100
- *Description* 100
- *Inspector* 101
- *District* 80
- *InspectionResultNumber* **50**

```
Proc sql ;
    Select
            max(length(IdentifierValue))
                label="longest IdentifierValue"
          , max(length(PrimaryEstablishmentNumber))
                label="longest PrimaryEstablishmentNumber"
          , max(length(EstablishmentName))
                label="longest EstablishmentName"
          , max(length(Description)) label="longest Description"
          , max(length(Inspector)) label="longest Inspector"
          , max(length(District)) label="longest District"
          , max(length(InspectionResultNumber))
                label="longest InspectionResultNumber"
        into
            :lenIdentifierValue
          , :lenPrimaryEstablishmentNumber
          , :lenEstablishmentName
          , :lenDescription
          , :lenInspector
          , :lenDistrict
          , :lenInspectionResultNumber
    from Inspection ;
Quit ;
```

longest Identifier Value	longest Primary Establishment Number	longest Establishment Name	longest Description	longest Inspector	longest District	longest Inspection ResultNumber
38	7	55	1	25	16	14

Appendix 7: Shorten the Character Variables in Inspection

```
/* The following step will produce the following warning for each of
   the character variables we are shortening.
       Multiple lengths were specified for the variable ... by input
       data set(s).  This may cause truncation of data.
   This is exactly what we want.  SAS provides no mechanism to turn off
   these warnings.  We could avoid them by renaming the variables in
   the SET statement, writing their values to new variables, and
   dropping the renamed variables.  But, that's a lot of code just to
   avoid the warnings. */
   Data Inspection( label="Inspections and Establishments" ) ;
       Length
           IdentifierValue            $&lenIdentifierValue
           EstablishmentId            8
           ShiftTypeId                8
           PrimaryEstablishmentNumber $&lenPrimaryEstablishmentNumber
           EstablishmentName          $&lenEstablishmentName
           DistrictNumber             $2
           InspectorId                8
           Description                $&lenDescription
           StartDate                  8
           Inspector                  $&lenInspector
           District                   $&lenDistrict
           InspectionResultNumber     $&lenInspectionResultNumber
       ;
       Set Inspection ;
    /* Remove all formats except for the StartDate. */
       Format
           IdentifierValue--Description
           Inspector--InspectionResultNumber
       ;
    /* Remove all inFormats. */
       InFormat _all_ ;
   Run ;
   %symDel
       lenIdentifierValue
       lenPrimaryEstablishmentNumber
       lenEstablishmentName
       lenDescription
       lenInspector
       lenDistrict
       lenInspectionResultNumber
   ;
```

Appendix 8: Query to Join Questionnaire and Inspection to Create the Country Dataset

```
Proc sql ;
    Create table Country as
        select
            Questionnaire.QuestionnaireDefID
          , Questionnaire.Name
          , Questionnaire.Title
          , Questionnaire.GrpSortOrd
          , Questionnaire.QuestionText
          , Questionnaire.QDefSortOrd
          , Questionnaire.QuestionType
          , Questionnaire.Answer
          , Questionnaire.QuestionDefID
          , Questionnaire.QuestionnaireID
          , Questionnaire.TaskConnector
          , Questionnaire.QuestionID
          , Inspection.PrimaryEstablishmentNumber
          , Inspection.EstablishmentName
          , Inspection.DistrictNumber
          , Inspection.Description
          , Inspection.StartDate
          , Inspection.Inspector
          , Inspection.District
          , Inspection.InspectionResultNumber
        from Questionnaire left join Inspection on (
            Questionnaire.TaskConnector eq Inspection.IdentifierValue
        )
        order by Questionnaire.GrpSortOrd , Questionnaire.QDefSortOrd
    ;
    Drop table
        Questionnaire
      , Inspection
    ;
Quit ;
```