Complex Data Combinations on Large Volumes: how to get the Best Performance
Henri THEUWISSEN, BI Knowledge Sharing, Belgium

ABSTRACT
Developers are often challenged to build complex combinations of large files with a good performance.
In most cases the SQL procedure is the obvious choice. However, the DATA step often provides better alternatives with outstanding performance improvements.
An extra level of complexity is added when data can not just be combined on common key fields, but time slices or weighting parameters have to be taken into account.

INTRODUCTION
We will learn how to build solutions with a good performance for complex combinations of tables with millions of records.

We will compare the following methods:
- SQL inner joins
- MERGE - BY statements
- hash tables
- arrays
- multiple SET statements

The following business cases will be discussed:
- How can we efficiently treat data in a star schema with a fact table and several dimension tables?
- How can we efficiently treat time-sliced data where every record must be split into several records using a weight variable from another table?
- How can we efficiently combine two time-sliced tables and search for gaps in the combination of these tables?

BUSINESS CASE 1
Combining a Fact Table with Several Dimension Tables

INPUT TABLES
We have data in a star schema with a fact table (PORTFOLIO) and several dimension tables (CLIENT, INSTALLATION, CONTRACT and INVOICING) with unique key values.
We will combine the data from the PORTFOLIO fact table with the CLIENT, INSTALLATION, CONTRACT and INVOICING dimension tables using the following methods:

- SQL inner joins
- DATA step MERGE - BY statements
- hash tables

The resulting table will look as follows:

**SOLUTION 1**

*Using SQL Inner Joins*

Combine all tables at once in one query using SQL inner joins.

```sql
PROC SQL;
CREATE TABLE BC1_SOL1 AS
SELECT POR.*, CLI.CLIENT_NAME, CON.SERVICE, INS.POSTAL_CODE, INV.ACCOUNT, INV.DIRECT_DEBIT
FROM PORTFOLIO POR, CLIENT CLI, CONTRACT CON, INSTALLATION INS, INVOICING INV
WHERE POR.INSTALLATION = INS.INSTALLATION AND POR.CONTRACT = CON.CONTRACT AND POR.CLIENT = CLI.CLIENT AND POR.CONTRACT = INV.CONTRACT;
QUIT;
```
SOLUTION 2

*Using DATA Step MERGE – BY Statements*

Combine the PORTFOLIO fact table with the INSTALLATION dimension table by INSTALLATION using a first DATA step MERGE statement.

```
PROC SORT DATA = PORTFOLIO;
   BY INSTALLATION;
RUN;

PROC SORT DATA = INSTALLATION;
   BY INSTALLATION;
RUN;

DATA PORTFOLIO_INSTALLATION;
   MERGE PORTFOLIO (IN = POR) INSTALLATION;
   BY INSTALLATION;
   IF POR;
RUN;
```

Combine the intermediate result with the CONTRACT and INVOICING dimension tables by CONTRACT using a second DATA step MERGE statement.

```
PROC SORT DATA = PORTFOLIO_INSTALLATION;
   BY CONTRACT;
RUN;

PROC SORT DATA = CONTRACT;
   BY CONTRACT;
RUN;

PROC SORT DATA = INVOICING;
   BY CONTRACT;
RUN;

DATA PORTFOLIO_INSTALLATION_CONTRACT;
   MERGE PORTFOLIO_INSTALLATION (IN = POR) CONTRACT INVOICING;
   BY CONTRACT;
   IF POR;
RUN;
```

Combine the intermediate result with the CLIENT dimension table by CLIENT using a third DATA step MERGE statement.

```
PROC SORT DATA = PORTFOLIO_INSTALLATION_CONTRACT;
   BY CLIENT;
RUN;

PROC SORT DATA = CLIENT;
   BY CLIENT;
RUN;

DATA BC1_SOL2;
   MERGE PORTFOLIO_INSTALLATION_CONTRACT (IN = POR) CLIENT;
   BY CLIENT;
   IF POR;
RUN;
```
COMBINE the PORTFOLIO fact table with all the dimension tables at once in one DATA step by loading the dimension tables into hash objects.

DATA BC1_SOL3;
SET PORTFOLIO;
LENGTH CLIENT_NAME  $ 40
    SERVICE      $  3
    POSTAL_CODE  $  4
    ACCOUNT      $  8
    DIRECT_DEBIT $  1;
IF _N_ = 1 THEN DO;
   DECLARE HASH CLI (DATASET : 'CLIENT');  
   CLI.DEFINEKEY ('CLIENT');
   CLI.DEFINEDATA ('CLIENT_NAME');
   CLI.DEFINEDONE ( );
   DECLARE HASH CON (DATASET : 'CONTRACT');  
   CON.DEFINEKEY ('CONTRACT');
   CON.DEFINEDATA ('SERVICE');
   CON.DEFINEDONE ( );
   DECLARE HASH INS (DATASET : 'INSTALLATION');  
   INS.DEFINEKEY ('INSTALLATION');
   INS.DEFINEDATA ('POSTAL_CODE');
   INS.DEFINEDONE ( );
   DECLARE HASH INV (DATASET : 'INVOICING');  
   INV.DEFINEKEY ('CONTRACT');
   INV.DEFINEDATA ('ACCOUNT', 'DIRECT_DEBIT');
   INV.DEFINEDONE ( );
END;
DROP RC;
RC = CLI.FIND ( );
RC = CON.FIND ( );
RC = INS.FIND ( );
RC = INV.FIND ( );
RUN;

1. Load the CLIENT table into a hash object using the CLIENT variable as the key and the CLIENT_NAME values as the data.
2. Load the CONTRACT table into a hash object using the CONTRACT variable as the key and the SERVICE values as the data.
3. Load the INSTALLATION table into a hash object using the INSTALLATION variable as the key and the POSTAL_CODE values as the data.
4. Load the INVOICING table into a hash object using the CONTRACT variable as the key and the ACCOUNT and DIRECT_DEBIT values as the data.
5. Look up the dimension values in the hash objects.
RESOURCE USAGE

The following tables and charts illustrate resource usage in terms of:

- real time
- user CPU
- memory

<table>
<thead>
<tr>
<th>REAL TIME</th>
<th>SQL</th>
<th>MERGE</th>
<th>HASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 × 1,000,000</td>
<td>54 s</td>
<td>72 s</td>
<td>12 s</td>
</tr>
<tr>
<td>3 × 2,000,000</td>
<td>113 s</td>
<td>149 s</td>
<td>21 s</td>
</tr>
<tr>
<td>3 × 3,000,000</td>
<td>140 s</td>
<td>175 s</td>
<td>21 s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USER CPU</th>
<th>SQL</th>
<th>MERGE</th>
<th>HASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 × 1,000,000</td>
<td>12 s</td>
<td>14 s</td>
<td>1 s</td>
</tr>
<tr>
<td>3 × 2,000,000</td>
<td>23 s</td>
<td>28 s</td>
<td>14 s</td>
</tr>
<tr>
<td>3 × 3,000,000</td>
<td>34 s</td>
<td>38 s</td>
<td>20 s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MEMORY</th>
<th>SQL</th>
<th>MERGE</th>
<th>HASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 × 1,000,000</td>
<td>74,016 KB</td>
<td>88,678 KB</td>
<td>389,478 KB</td>
</tr>
<tr>
<td>3 × 2,000,000</td>
<td>74,043 KB</td>
<td>88,878 KB</td>
<td>492,972 KB</td>
</tr>
<tr>
<td>3 × 3,000,000</td>
<td>74,171 KB</td>
<td>89,078 KB</td>
<td>615,288 KB</td>
</tr>
</tbody>
</table>
BUSINESS CASE 2
Splitting Time-sliced Data Using a Weight Variable

INPUT TABLES

The METERING table contains the CONSUMPTION measured for an INSTALLATION between 2 dates (MET_FROM and MET_TO).

The SLP_DAY table contains the weight (SLP) for every day (DATE).

We have to split the CONSUMPTION from the METERING table into records per year using the weight variable SLP from the SLP_DAY table.

We will split the CONSUMPTION per year using the following steps:

1. Make the daily SLP values available using one of the following methods:
   - SQL joins
   - a hash table
   - an array

2. Calculate the total SLP value for the consumption period (TOTAL_SLP) using the following formula:

   \[
   TOTAL\_SLP = \sum_{DATE = MET\_FROM}^{MET\_TO} SLP
   \]

3. Calculate the total yearly SLP values (YEAR_SLP) using the following formulas:

   \[
   YEAR\_SLP = \sum_{DATE = MET\_FROM}^{MET\_TO} SLP
   \]

4. Split the CONSUMPTION value per year (CON_YEAR + CON_VALUE) using the following formula:

   \[
   CON\_VALUE = YEAR\_SLP / TOTAL\_SLP \times CONSUMPTION
   \]

The resulting table will look as follows:
The following table shows how the CONSUMPTION for INSTALLATION INST_002 is split across 2009 and 2010:

<table>
<thead>
<tr>
<th>INSTALLATION</th>
<th>MET_FROM</th>
<th>MET_TO</th>
<th>CONSUMPTION</th>
<th>CON_YEAR</th>
<th>CON_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST_001</td>
<td>23/01/2009</td>
<td>30/05/2009</td>
<td>0.00</td>
<td>2009</td>
<td>0.00</td>
</tr>
<tr>
<td>INST_002</td>
<td>18/01/2009</td>
<td>16/01/2010</td>
<td>5,000</td>
<td>2009</td>
<td>4,729.16</td>
</tr>
<tr>
<td>INST_002</td>
<td>18/01/2009</td>
<td>16/01/2010</td>
<td>5,000</td>
<td>2010</td>
<td>280.84</td>
</tr>
<tr>
<td>INST_003</td>
<td>12/01/2009</td>
<td>11/01/2010</td>
<td>1,000</td>
<td>2009</td>
<td>650.22</td>
</tr>
<tr>
<td>INST_003</td>
<td>12/01/2009</td>
<td>11/01/2010</td>
<td>1,000</td>
<td>2010</td>
<td>349.79</td>
</tr>
</tbody>
</table>

SOLUTION 1
Using SQL Joins

Use traditional SQL processing. Three PROC SQL steps will be required. The execution of the first two queries involves performing a Cartesian product join that can not be optimized.

Calculate the total SLP value for the consumption period.

PROC SQL;
CREATE TABLE BC2_SOL1A AS
SELECT INSTALLATION, MET_FROM, MET_TO, CONSUMPTION, SUM(SLP) AS TOTAL_SLP
FROM METERING, SLP_DAY
WHERE DATE BETWEEN MET_FROM AND MET_TO
GROUP BY INSTALLATION, MET_FROM, MET_TO, CONSUMPTION;
QUIT;

NOTE: The execution of this query involves performing one or more Cartesian product joins that can not be optimized.
Calculate the total yearly SLP values.

PROC SQL;
CREATE TABLE BC2_SOL1B AS
SELECT INSTALLATION,
       YEAR (DATE) AS CON_YEAR,
       MET_FROM,
       MET_TO,
       SUM (SLP) AS YEAR_SLP
FROM METERING,
SLP_DAY
WHERE DATE BETWEEN MET_FROM AND MET_TO
GROUP BY INSTALLATION,
       CALCULATED CON_YEAR,
       MET_FROM,
       MET_TO;
QUIT;

NOTE: The execution of this query involves performing one or more Cartesian product joins that can not be optimized.

Split the CONSUMPTION value per year.

PROC SQL;
CREATE TABLE BC2_SOL1 (DROP = TOTAL_SLP YEAR_SLP) AS
SELECT A.*,
       CON_YEAR,
       YEAR_SLP,
       CONSUMPTION * YEAR_SLP / TOTAL_SLP AS CON_VALUE
FROM BC2_SOL1A A,
BC2_SOL1B B
WHERE A.INSTALLATION = B.INSTALLATION AND
A.MET_FROM = B.MET_FROM AND
A.MET_TO = B.MET_TO;
QUIT;
SOLUTION 2  
Using a Hash Table

Load the SLP values into a hash table.

```plaintext
DATA BC2_SOL2 (KEEP = INSTALLATION MET_FROM MET_TO CONSUMPTION CON_YEAR CON_VALUE);
SET METERING;
DROP RC;
IF _N_ = 1 THEN DO;
DECLARE HASH HT (DATASET : 'SLP_DAY');
HT.DEFINEKEY ('DATE');
HT.DEFINEDATA ('SLP');
HT.DEFINEDONE();
END;
TOTAL_SLP = 0;
DO DATE = MET_FROM TO MET_TO;
RC = HT.FIND();
IF RC = 0 THEN TOTAL_SLP = TOTAL_SLP + SLP;
END;
CON_VALUE = 0;
YEAR_SLP = 0;
DO DATE = MET_FROM TO MET_TO;
RC = HT.FIND();
IF RC = 0 THEN YEAR_SLP = YEAR_SLP + SLP;
IF DATE = MET_TO THEN DO;
CON_VALUE = YEAR_SLP / TOTAL_SLP * CONSUMPTION;
CON_YEAR = YEAR (DATE);
OUTPUT;
END;
ELSE DO;
IF YEAR (DATE) NE YEAR (DATE + 1) THEN DO;
CON_VALUE = YEAR_SLP / TOTAL_SLP * CONSUMPTION;
CON_YEAR = YEAR (DATE);
OUTPUT;
YEAR_SLP = 0;
END;
END;
RUN;
```

1. Load the SLP_DAY table into a hash object using the DATE variable as the key and the SLP values as the data.
2. Look up the SLP values for the consumption period in the hash object.
3. Calculate the total SLP value for the consumption period.
4. Calculate the total yearly SLP values.
5. Split the CONSUMPTION value per year.
SOLUTION 3
Using an Array

Perform a table lookup in an array that contains the SLP values.

```sas
PROC SQL NOPRINT;
SELECT MIN (DATE),
MAX (DATE)
INTO :ARR_START,
:ARR_STOP
FROM SLP_DAY;
QUIT;

%LET ARR_START = &ARR_START;
%LET ARR_STOP  = &ARR_STOP;
%LET ARR_RANGE = &ARR_START : &ARR_STOP;

DATA BC2_SOL3 (KEEP = INSTALLATION MET_FROM MET_TO CONSUMPTION CON_YEAR CON_VALUE);
SET METERING;
FORMAT CON_VALUE COMMA12.2;
ARRAY SLP_VAL (&ARR_RANGE) SLP_VAL_&ARR_START - SLP_VAL_&ARR_STOP;
RETAIN SLP_VAL_&ARR_START - SLP_VAL_&ARR_STOP;
IF _N_ = 1 THEN DO;
DO INDEX1 = 1 TO NOBS;
SET SLP_DAY POINT = INDEX1 NOBS = NOBS;
SLP_VAL (DATE) = SLP;
END;
END;
TOTAL_SLP = 0;
DO INDEX2 = MET_FROM TO MET_TO;
TOTAL_SLP = TOTAL_SLP + SLP_VAL (INDEX2);
END;
CON_VALUE = 0;
YEAR_SLP = 0;
DO INDEX3 = MET_FROM TO MET_TO;
YEAR_SLP = YEAR_SLP + SLP_VAL (INDEX3);
IF INDEX3 = MET_TO THEN DO;
CON_VALUE = YEAR_SLP / TOTAL_SLP * CONSUMPTION;
CON_YEAR = YEAR (INDEX3);
OUTPUT;
END;
ELSE DO;
IF YEAR (INDEX3) NE YEAR (INDEX3 + 1) THEN DO;
CON_VALUE = YEAR_SLP / TOTAL_SLP * CONSUMPTION;
CON_YEAR = YEAR (INDEX3);
OUTPUT;
YEAR_SLP = 0;
END;
END;
END;
RUN;
```
11

- Create macro variables to specify the index range of the array.

<table>
<thead>
<tr>
<th>Macro Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARR_START</td>
<td>01/01/2009</td>
</tr>
<tr>
<td>ARR_STOP</td>
<td>31/12/2010</td>
</tr>
<tr>
<td>ARR_RANGE</td>
<td>1/01/2009:31/12/2010</td>
</tr>
</tbody>
</table>

- Define the array to contain the daily SLP values.

<table>
<thead>
<tr>
<th>Array Name</th>
<th>SLRP VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Element</td>
<td>1 17900</td>
</tr>
<tr>
<td></td>
<td>2 17900</td>
</tr>
<tr>
<td></td>
<td>3 179000</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Array Reference</td>
<td>SLRP VAL 17900</td>
</tr>
<tr>
<td></td>
<td>SLRP VAL 179000</td>
</tr>
<tr>
<td></td>
<td>SLRP VAL 179000</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Variable</td>
<td>SLRP VAL 17900</td>
</tr>
<tr>
<td></td>
<td>SLRP VAL 179000</td>
</tr>
<tr>
<td></td>
<td>SLRP VAL 179000</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Value</td>
<td>0.0032458222</td>
</tr>
<tr>
<td></td>
<td>0.0032458256</td>
</tr>
<tr>
<td></td>
<td>0.0032458772</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

- Process the array to create the SLP VAL variables containing the daily SLP values.

| PDV | SLRP VAL 17900 | SLRP VAL 179000 | SLRP VAL 1790000 | TOTAL SLRP | N |
|-----|---------------|-----------------|------------------|------------|
|     | 0.0032458222  | 0.0032458256    | 0.0032458772     | 0.003245893 | 0 | 1 |

- Calculate the total SLP value for the consumption period.

<table>
<thead>
<tr>
<th>PDV</th>
<th>INSTALLATION</th>
<th>MET FROM</th>
<th>MET TO</th>
<th>TOTAL SLRP</th>
<th>INDEX2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INST 001</td>
<td>21/08/2009</td>
<td>...</td>
<td>0 + 0.003245822 = 0.0032458225</td>
<td>01130</td>
</tr>
<tr>
<td></td>
<td>INST 001</td>
<td>21/08/2009</td>
<td>...</td>
<td>...</td>
<td>18131</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>INST 001</td>
<td>21/08/2009</td>
<td>...</td>
<td>0.0032458225 + 0.00324589335 = 0.0064915800</td>
<td>01132</td>
</tr>
</tbody>
</table>

- Calculate the total yearly SLP values.

- Split the CONSUMPTION value per year.
RESOURCES USAGE

The following tables and charts illustrate resource usage in terms of:

- real time
- user CPU
- memory

### Real Time

<table>
<thead>
<tr>
<th>Resource</th>
<th>SQL</th>
<th>HASH</th>
<th>ARRAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000,000</td>
<td>1,155 s</td>
<td>112 s</td>
<td>57 s</td>
</tr>
<tr>
<td>3,000,000</td>
<td>8,081 s</td>
<td>398 s</td>
<td>201 s</td>
</tr>
<tr>
<td>5,000,000</td>
<td>13,564 s</td>
<td>623 s</td>
<td>313 s</td>
</tr>
</tbody>
</table>

### User CPU

<table>
<thead>
<tr>
<th>Resource</th>
<th>SQL</th>
<th>HASH</th>
<th>ARRAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000,000</td>
<td>563 s</td>
<td>111 s</td>
<td>56 s</td>
</tr>
<tr>
<td>3,000,000</td>
<td>2,944 s</td>
<td>389 s</td>
<td>199 s</td>
</tr>
<tr>
<td>5,000,000</td>
<td>4,261 s</td>
<td>621 s</td>
<td>310 s</td>
</tr>
</tbody>
</table>

### Memory

<table>
<thead>
<tr>
<th>Resource</th>
<th>SQL</th>
<th>HASH</th>
<th>ARRAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000,000</td>
<td>73,628 kB</td>
<td>828 kB</td>
<td>484 MB</td>
</tr>
<tr>
<td>3,000,000</td>
<td>73,628 kB</td>
<td>852 kB</td>
<td>474 MB</td>
</tr>
<tr>
<td>5,000,000</td>
<td>73,628 kB</td>
<td>836 kB</td>
<td>480 MB</td>
</tr>
</tbody>
</table>
BUSINESS CASE 3
Combining Two Time-sliced Tables

INPUT TABLES

The CONTRACT table contains the period (CON_FROM – CON_TO) that an INSTALLATION is active.

The METERING table contains the CONSUMPTION measured for an INSTALLATION between 2 dates (MET_FROM and MET_TO).

We have to search for the periods in 2009 that although an INSTALLATION is active no CONSUMPTION has been measured.

<table>
<thead>
<tr>
<th>INSTALLATION</th>
<th>CON_FROM</th>
<th>CON_TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST_001</td>
<td>01/07/2008</td>
<td>31/12/2008</td>
</tr>
<tr>
<td>INST_002</td>
<td>01/07/2008</td>
<td>31/12/2008</td>
</tr>
<tr>
<td>INST_003</td>
<td>01/1/2008</td>
<td>31/12/2008</td>
</tr>
<tr>
<td>INST_004</td>
<td>01/07/2008</td>
<td>31/12/2008</td>
</tr>
</tbody>
</table>

The following table shows the periods in 2009 that although an INSTALLATION is active no CONSUMPTION has been measured:

<table>
<thead>
<tr>
<th>INSTALLATION</th>
<th>MET_FROM</th>
<th>MET_TO</th>
<th>CONSUMPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST_001</td>
<td>01/10/2009</td>
<td>31/03/2009</td>
<td>100</td>
</tr>
<tr>
<td>INST_001</td>
<td>01/04/2009</td>
<td>31/05/2009</td>
<td>80</td>
</tr>
<tr>
<td>INST_001</td>
<td>01/10/2009</td>
<td>31/03/2010</td>
<td>150</td>
</tr>
<tr>
<td>INST_002</td>
<td>01/07/2009</td>
<td>31/10/2009</td>
<td>1,000</td>
</tr>
<tr>
<td>INST_002</td>
<td>01/09/2009</td>
<td>31/06/2009</td>
<td>2,000</td>
</tr>
<tr>
<td>INST_002</td>
<td>01/11/2009</td>
<td>31/11/2009</td>
<td>2,500</td>
</tr>
<tr>
<td>INST_003</td>
<td>01/12/2008</td>
<td>31/08/2008</td>
<td>1,000</td>
</tr>
<tr>
<td>INST_003</td>
<td>01/07/2009</td>
<td>31/10/2009</td>
<td>700</td>
</tr>
<tr>
<td>INST_004</td>
<td>01/09/2009</td>
<td>31/10/2009</td>
<td>800</td>
</tr>
</tbody>
</table>

The resulting table will look as follows:

<table>
<thead>
<tr>
<th>INSTALLATION</th>
<th>UNM_FROM</th>
<th>UNM_TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST_002</td>
<td>01/08/2009</td>
<td>31/08/2009</td>
</tr>
<tr>
<td>INST_002</td>
<td>01/11/2009</td>
<td>31/02/2009</td>
</tr>
<tr>
<td>INST_002</td>
<td>01/12/2009</td>
<td>31/02/2009</td>
</tr>
<tr>
<td>INST_003</td>
<td>01/11/2009</td>
<td>31/02/2009</td>
</tr>
<tr>
<td>INST_004</td>
<td>01/07/2009</td>
<td>31/07/2009</td>
</tr>
<tr>
<td>INST_004</td>
<td>01/11/2009</td>
<td>31/02/2009</td>
</tr>
</tbody>
</table>
SOLUTION 1
Using the DATA Step MERGE – BY Statement

Use DATA step MERGE – BY statements to search for the periods in 2009 that although an INSTALLATION is active no CONSUMPTION has been measured.

DATA _NULL_; ①
  CALL SYMPUT ('START_RANGE', TRIM (LEFT ('01JAN2009'D))); ②
  CALL SYMPUT ('END_RANGE', TRIM (LEFT ('31DEC2009'D))); ③
RUN;

PROC SORT DATA = CONTRACT;
  BY INSTALLATION CON_FROM;
RUN;

DATA CON_DAY (KEEP = INSTALLATION DATE); ④
  SET CONTRACT;
  DO DATE = MAX (&START_RANGE, CON_FROM) TO MIN (&END_RANGE, CON_TO);
  OUTPUT;
  END;
RUN;

PROC SORT DATA = METERING;
  BY INSTALLATION MET_FROM;
RUN;

DATA MET_DAY (KEEP = INSTALLATION DATE); ⑤
  SET METERING;
  DO DATE = MAX (&START_RANGE, MET_FROM) TO MIN (&END_RANGE, MET_TO);
  OUTPUT;
  END;
RUN;

DATA BC3_SOL1 (KEEP = INSTALLATION UNM_FROM UNM_TO);
  MERGE CON_DAY (IN = C) MET_DAY (IN = M);
  BY INSTALLATION DATE;
  IF C AND NOT M; ⑥
  RETAIN UNM_FROM PREV_DATE .;
  FORMAT UNM_FROM UNM_TO DDMMYY10.; ⑦
  IF FIRST.INSTALLATION THEN DO;
    UNM_FROM = DATE;
    PREV_DATE = DATE;
  END;
  IF DATE - PREV_DATE GT 1 THEN DO;
    UNM_TO = PREV_DATE;
    OUTPUT; ⑧
    UNM_FROM = DATE;
  END;
  IF LAST.INSTALLATION THEN DO;
    UNM_TO = DATE;
    OUTPUT; ⑨
  END;
  PREV_DATE = DATE;
RUN;

① Create macro variables to restrict the analysis to 2009.

<table>
<thead>
<tr>
<th>MACRO VARIABLE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>START_RANGE</td>
<td>17898</td>
</tr>
<tr>
<td>END_RANGE</td>
<td>31122009</td>
</tr>
</tbody>
</table>

14
Split the CONTRACT table per day in 2009.

Split the METERING table per day in 2009.

Merge the CON_DAY and MET_DAY tables by INSTALLATION and DATE.

Only keep the observations from the CON_DAY table that do not exist in the MET_DAY table.

Output only one observation per period (UNM_FROM – UNM_TO).
SOLUTION 2
Using Arrays

Use arrays to search for the periods in 2009 that although an INSTALLATION is active no CONSUMPTION has been measured.

DATA _NULL_; ①
   CALL SYMPUT ('START_RANGE', TRIM (LEFT ('01JAN2009'D))); ②
   CALL SYMPUT ('END_RANGE', TRIM (LEFT ('31DEC2009'D))); ③
RUN;

DATA BC3_SOL2 (KEEP = INSTALLATION UNM_FROM UNM_TO); ④
   MERGE CONTRACT (IN = C) METERING;
   BY INSTALLATION;
   FORMAT UNM_FROM UNM_TO DDMMYY10.; ⑤
   RETAIN CON_&START_RANGE - CON_&END_RANGE ' ' ⑥
       MET_&START_RANGE - MET_&END_RANGE ' ' ⑦
       UNM_&START_RANGE - UNM_&END_RANGE;
   ARRAY CON (&START_RANGE : &END_RANGE) $ 1 CON_&START_RANGE - CON_&END_RANGE; ⑧
   ARRAY MET (&START_RANGE : &END_RANGE) $ 1 MET_&START_RANGE - MET_&END_RANGE; ⑨
   ARRAY UNM (&START_RANGE : &END_RANGE) $ 1 UNM_&START_RANGE - UNM_&END_RANGE; ⑩
IF FIRST.INSTALLATION THEN DO;
   DO IR = &START_RANGE TO &END_RANGE;
      CON (IR) = '-'; ⑪
      MET (IR) = '-'; ⑫
      UNM (IR) = '-'; ⑬
   END;
   DO IC = MAX (&START_RANGE, CON_FROM) TO MIN (&END_RANGE, CON_TO);
      CON (IC) = 'X'; ⑭
   END;
   DO IC = MAX (&START_RANGE, MET_FROM) TO MIN (&END_RANGE, MET_TO);
      MET (IC) = 'X'; ⑮
   END;
IF LAST.INSTALLATION THEN DO;
   DO IC = &START_RANGE TO &END_RANGE;
      IF CON (IC) = 'X' AND MET (IC) NE 'X' THEN UNM (IC) = 'X'; ⑯
   END;
   DO IC = &START_RANGE TO &END_RANGE;
      IF UNM (IC) = 'X' THEN DO;
         IF (IC NE &START_RANGE AND UNM (IC - 1) = '-') ⑰
            OR IC = &START_RANGE THEN UNM_FROM = IC;
         IF (IC NE &END_RANGE AND UNM (IC + 1) = '-') ⑱
            OR (IC = &END_RANGE THEN DO;
               UNM_TO = IC;
               OUTPUT; ⑲
            END;
      END;
   END;
RUN;
1. Create macro variables to restrict the analysis to 2009.

<table>
<thead>
<tr>
<th>MACRO SYMBOL TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIABLE</td>
</tr>
<tr>
<td>START_RANGE</td>
</tr>
<tr>
<td>END_RANGE</td>
</tr>
</tbody>
</table>

2. Merge the CONTRACT and METERING tables by INSTALLATION.

<table>
<thead>
<tr>
<th>INSTALLATION</th>
<th>CON_FROM</th>
<th>CON_TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST_001</td>
<td>01/07/2008</td>
<td>30/04/2009</td>
</tr>
<tr>
<td>INST_002</td>
<td>01/07/2009</td>
<td>31/12/2010</td>
</tr>
<tr>
<td>INST_003</td>
<td>01/12/2003</td>
<td>31/09/2009</td>
</tr>
<tr>
<td>INST_004</td>
<td>01/07/2009</td>
<td>31/12/2010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INSTALLATION</th>
<th>METER</th>
<th>MET_TO</th>
<th>CONSUMPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST_001</td>
<td>01/01/2009</td>
<td>31/03/2009</td>
<td>100</td>
</tr>
<tr>
<td>INST_002</td>
<td>01/07/2010</td>
<td>31/03/2010</td>
<td>150</td>
</tr>
<tr>
<td>INST_003</td>
<td>01/07/2009</td>
<td>30/09/2009</td>
<td>1000</td>
</tr>
<tr>
<td>INST_004</td>
<td>01/07/2009</td>
<td>31/10/2009</td>
<td>2000</td>
</tr>
</tbody>
</table>

3. Define the array CON based on the CONTRACT table to contain the days in 2009 that an INSTALLATION is active.

4. Define the array MET based on the METERING table to contain the days in 2009 that CONSUMPTION has been measured.

5. Define the array UNM to contain the days in 2009 that although an INSTALLATION is active no CONSUMPTION has been measured.
Process the CON, MET and UNM arrays to initialize the CON, MET and UNM variables to "-" for each INSTALLATION.

**Process the CON array to assign the value "X" to the CON variables for the days in 2009 that an INSTALLATION is active.**

Process the MET array to assign the value "X" to the MET variables for the days in 2009 that CONSUMPTION has been measured.
Process the CON, MET and UNM arrays to assign the value "X" to the UNM variables for the days in 2009 that although an INSTALLATION is active no CONSUMPTION has been measured.

Output only one observation per period (UNM_FROM – UNM_TO).
SOLUTION 3

Using Multiple SET Statements

Use multiple SET statements to search for the periods in 2009 that although an INSTALLATION is active no CONSUMPTION has been measured.

DATA _NULL_; ①
   CALL SYMPUT('START_RANGE', TRIM(LEFT('01JAN2009'D)));
   CALL SYMPUT('END_RANGE', TRIM(LEFT('31DEC2009'D)));
RUN;

DATA BC3_SOL3 (KEEP = INSTALLATION UNM_FROM UNM_TO);
   SET CONTRACT; ②
   FORMAT UNM_FROM UNM_TO DDMMYY10.;
   RETAIN OBS 0;
   CON_FROM = MAX(&START_RANGE, CON_FROM);
   CON_TO = MIN(&END_RANGE, CON_TO);
   FLAG = 0;
   DO WHILE (FLAG = 0 AND OBS LT NOBS);
      IF INSTALLATION GT MET_INSTALLATION
         OR (INSTALLATION = MET_INSTALLATION AND CON_FROM GT MET_TO) THEN DO;
         OBS = OBS + 1;
         SET METERING (RENAME = (INSTALLATION = MET_INSTALLATION)) ③
         POINT = OBS NOBS = NOBS;
         MET_FROM = MAX(&START_RANGE, MET_FROM);
         MET_TO = MIN(&END_RANGE, MET_TO);
      END;
      IF MET_INSTALLATION GT INSTALLATION
         OR (MET_INSTALLATION = INSTALLATION AND MET_FROM GT CON_FROM) THEN DO;
            UNM_FROM = CON_FROM;
            IF MET_INSTALLATION = INSTALLATION THEN UNM_TO = MIN(CON_TO, MET_FROM - 1);
            ELSE UNM_TO = CON_TO;
            OUTPUT;
            IF MET_INSTALLATION GT INSTALLATION THEN FLAG = 1;
      END;
      IF FLAG = 0 AND INSTALLATION = MET_INSTALLATION THEN DO; ④
         IF CON_TO GT MET_TO THEN DO;
            IF CON_FROM LE MET_TO THEN CON_FROM = MET_TO + 1;
         END;
         ELSE FLAG = 1;
      END;
   END;
   IF OBS = NOBS AND FLAG = 0 THEN DO; ⑤
      IF INSTALLATION = MET_INSTALLATION THEN DO;
         IF MET_FROM GT CON_FROM THEN DO;
            UNM_FROM = CON_FROM;
            UNM_TO = MET_FROM - 1;
         OUTPUT;
         CON_FROM = MET_FROM;
      END;
      IF MET_TO LT CON_TO THEN DO;
         UNM_FROM = MET_TO + 1;
         UNM_TO = CON_TO;
      OUTPUT;
   END;
   END;
ELSE DO;
   UNM_FROM = CON_FROM;
   UNM_TO = CON_TO;
   OUTPUT;
END;
END;
RUN;
1. Create macro variables to restrict the analysis to 2009.
2. Read the CONTRACT table sequentially.
3. Read the METERING table using direct access.
4. Use complex conditional processing to search for days in 2009 that although an INSTALLATION is active no CONSUMPTION has been measured.

**RESOURCE USAGE**
The following tables and charts illustrate resource usage in terms of:

- real time
- user CPU
- memory

<table>
<thead>
<tr>
<th>REAL TIME</th>
<th>SPLIT</th>
<th>ARRAY</th>
<th>SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>650,000 + 1,000,000</td>
<td>976 s</td>
<td>44 s</td>
<td>1 s</td>
</tr>
<tr>
<td>1,250,000 + 2,150,000</td>
<td>2,587 s</td>
<td>94 s</td>
<td>2 s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USER CPU</th>
<th>SPLIT</th>
<th>ARRAY</th>
<th>SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>650,000 + 1,000,000</td>
<td>191 s</td>
<td>43 s</td>
<td>1 s</td>
</tr>
<tr>
<td>1,250,000 + 2,150,000</td>
<td>495 s</td>
<td>93 s</td>
<td>2 s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MEMORY</th>
<th>SPLIT</th>
<th>ARRAY</th>
<th>SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>650,000 + 1,000,000</td>
<td>257 kB</td>
<td>582 kB</td>
<td>305 kB</td>
</tr>
<tr>
<td>1,250,000 + 2,150,000</td>
<td>258 kB</td>
<td>583 kB</td>
<td>305 kB</td>
</tr>
</tbody>
</table>
CONCLUSION

You should consider spending more programmer time to reduce computer resource usage for complex combinations of tables with millions of records. So, instead of using the traditional programming techniques like SQL inner joins or DATA step MERGE – BY statements, you should consider using more complex programming techniques like hash tables, arrays and multiple SET statements.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Henri THEUWISSEN
Enterprise: BI Knowledge Sharing
Address: Sterrenlaan 40
3360 BIERBEEK
BELGIUM
Work Phone: +32 (0)496 28 45 28
Fax: +32 (0)16 46 37 74
E-mail: henri.theuwissen@biknowledgesharing.be

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.

Other brand and product names are trademarks of their respective companies.