

Why Does My Job Run Soooo Slooowly..?

by Philip R Holland, Holland Numerics Ltd

Abstract

Almost every operation you are able to perform using SAS® software can be achieved in many different ways, each with their own advantages and disadvantages. It is not always possible to predict the disadvantages, as a basic operation may be suitable in one circumstance, but may be totally inappropriate in another. Using SAS/ACCESS® software and PROC SQL there are a wide variety of alternative methods, but, in general, only one is likely to give a solution and a fast response time. This paper explores these choices and gives an insight into how to determine the most efficient approach.

1. Introduction to SAS/ACCESS® software

The SAS/ACCESS component of SAS software offers facilities to access data in relational databases (e.g. DB2, Oracle, Informix). In this paper I will be concentrating on access to DB2 tables, but most of the examples could be applied in a similar way to other relational database systems.

I would like to consider, in particular, the choice to be made between Access Descriptors and Pass-Through SQL, and how much processing should be done in DB2, when designing the interface between SAS and DB2.

1.1. Access Descriptors vs Pass-Through SQL

Advantages of Access Descriptors:

- Can be used to update, insert and delete data in DB2 tables.
- Provides a simple interface to SAS, which makes the DB2 table look like a standard SAS dataset.
- Can be used to perform DB2 queries which have been pre-defined.

Disadvantages of Access Descriptors:

- When using an Access Descriptor a copy of all the DB2 data described in the Access Descriptor is transferred to SAS before any processing can be performed by the SAS software.
- DB2 does not perform any optimisation of the query defined by the Access Descriptor, i.e. there is no use made of any DB2 indexes.
- Cannot be used to perform DB2 queries which have not been pre-defined.

Advantages of Pass-Through SQL:

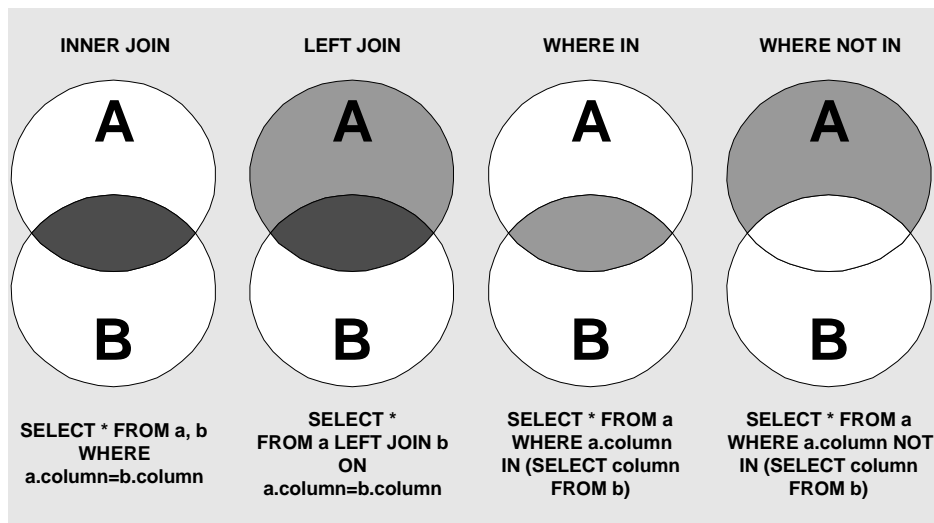
- Can be used to perform DB2 queries which have not been pre-defined.
- DB2 can optimise the query to use DB2 indexes.
- Can be used to execute any DB2 SQL statements, including the updating, inserting and deleting of data in DB2 tables, and creation, modification and deleting of DB2 tables. Provided the user has permission to perform these actions.

Disadvantages of Pass-Through SQL:

- DB2 SQL syntax must be understood.

2. PROC SQL

This paper is not intended as a tutorial on PROC SQL, but the following SQL constructions need to be explained here to clarify the discussion later in the paper.

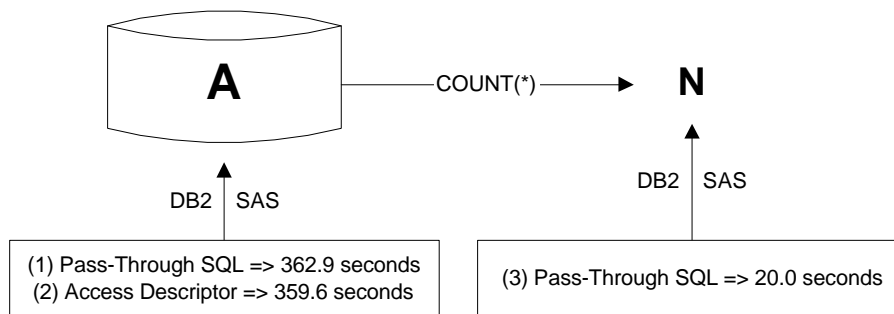


The SQL syntax used in this paper is compatible with both PROC SQL within SAS and in DB2 version 4.

3. Processing Data in DB2 or in SAS?

3.1. Summarising Rows in a DB2 Table

This example uses data from a single DB2 table, A, which has the rows counted using the summary function COUNT(*). The table A has 517,000 rows.



Sample code for the above flowchart:

(1) Pass-Through SQL:

```

SELECT COUNT(*)
FROM CONNECTION TO db2
  (SELECT *
   FROM a);
  
```

(2) Access Descriptor:

```
CREATE VIEW work.a AS
  SELECT *
  FROM CONNECTION TO db2
    (SELECT *
     FROM a); * or an Access Descriptor *;

SELECT COUNT(*)
FROM work.a;
```

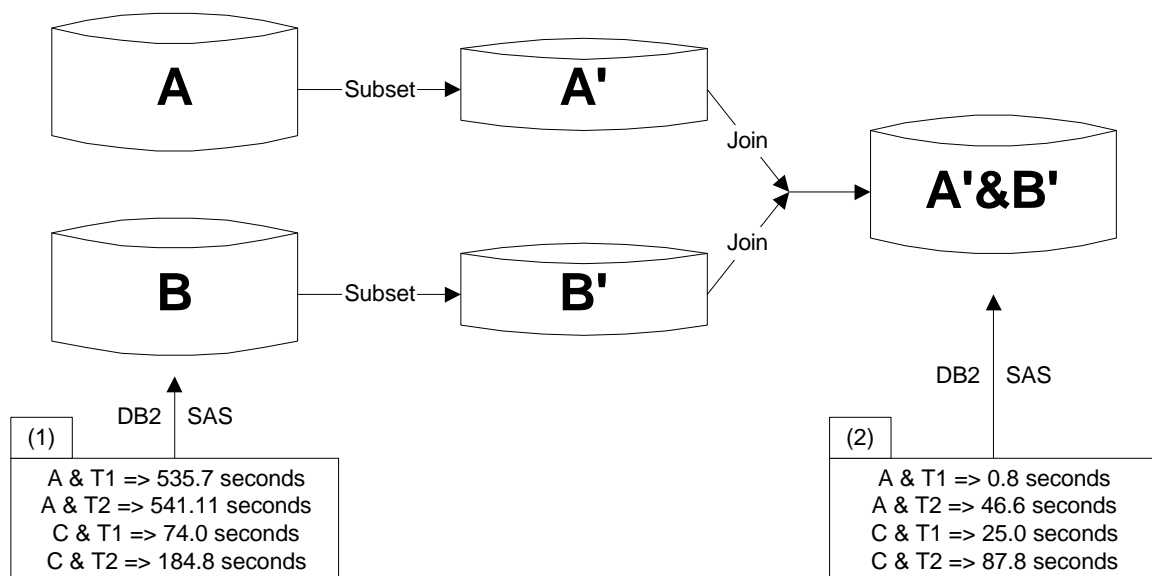
(3) Pass-Through SQL:

```
SELECT *
FROM CONNECTION TO db2
  (SELECT COUNT(*)
   FROM a);
```

3.2. Combining Subsets of 2 DB2 Tables

This example uses data from 2 DB2 tables of different sizes, one larger than the other. The tables will be joined using a column on each table which is indexed in DB2, and then the joined table is subsetted with a Where clause. The designing of a query on several DB2 tables from SAS needs to include consideration of when the data will be transferred to the SAS environment for manipulation by the SAS software.

There were 4 DB2 tables used to measure the following results: A (517,000 rows), C (120,000 rows), T1 (20,000 rows), and T2 (98,000 rows).



Sample code for the above flowchart:

(1) Access Descriptor:

```
CREATE VIEW work.t1 AS
  SELECT *
  FROM CONNECTION TO db2
    (SELECT *
     FROM t1); * or an Access Descriptor *;
```

```

CREATE VIEW work.c AS
  SELECT *
  FROM CONNECTION TO db2
      (SELECT *
       FROM c); * or an Access Descriptor *;

SELECT *
FROM work.t1
,work.c
WHERE t1.account=c.account
      AND t1.time=c.time
      AND c.trans IN ('0123','4567','8901')
      AND c.code_cf = 'C'
ORDER BY
      t1.account
      ,t1.number;

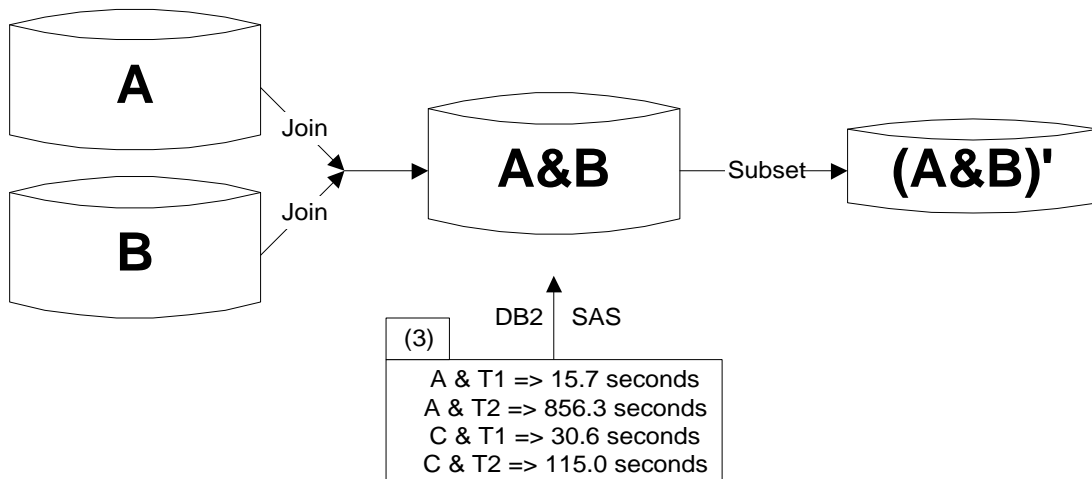
```

(2) Pass-Through SQL:

```

SELECT *
FROM CONNECTION TO db2
  (SELECT t1.*
      ,c.time AS ctime
  FROM t1
      ,c
  WHERE t1.account=c.account
        AND t1.time=c.time
        AND c.trans IN ('0123','4567','8901')
        AND c.code_cf = 'C') AS d
ORDER BY
      d.account
      ,d.number;

```



Sample code for the above flowchart:

(3) Pass-Through SQL:

```

SELECT *
FROM CONNECTION TO db2
  (SELECT t1.*
      ,c.time AS ctime

```

```

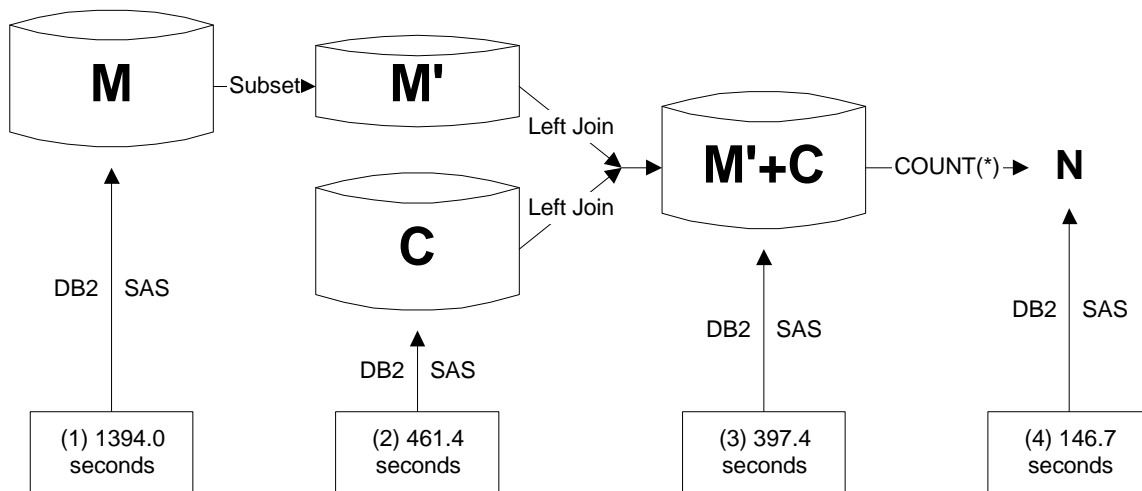
FROM t1
,c
WHERE t1.account=c.account
AND t1.time=c.time) AS d
WHERE d.trans IN ('0123','4567','8901')
AND d.code_cf = 'C'
ORDER BY
d.account
,d.number;

```

3.3. Adding Extra Information to a DB2 Table from another DB2 Table

This example uses data from 2 DB2 tables indexed using the same columns. To add additional information to the 1st table, where there may or may not be a matching record in the 2nd table, it is best to use the Left Join, preferably making use of indexed columns for matching the 2 tables.

There were 2 DB2 tables used to measure the following results: M (originally 5,200,000 rows and 575,000 rows selected), and C (640,000 rows).



Sample code for the above flowchart:

(1) Access Descriptor:

```

CREATE VIEW work.m AS
SELECT *
FROM CONNECTION TO db2
(SELECT *
FROM m); * or an Access Descriptor *;

CREATE VIEW work.c AS
SELECT *
FROM CONNECTION TO db2
(SELECT *
FROM c); * or an Access Descriptor *;

SELECT COUNT(*)
FROM (SELECT *
FROM work.m
WHERE SUBSTR(code,1,1) = 'A') AS m
LEFT JOIN
work.c

```

```
ON      m.account=c.account
AND    m.time=c.time;
```

(2) Pass-Through SQL:

```
CREATE VIEW work.m AS
SELECT *
FROM   CONNECTION TO db2
      (SELECT *
       FROM   m
       WHERE  SUBSTR(code,1,1) = 'A');
```

```
CREATE VIEW work.c AS
SELECT *
FROM   CONNECTION TO db2
      (SELECT *
       FROM   c);
```

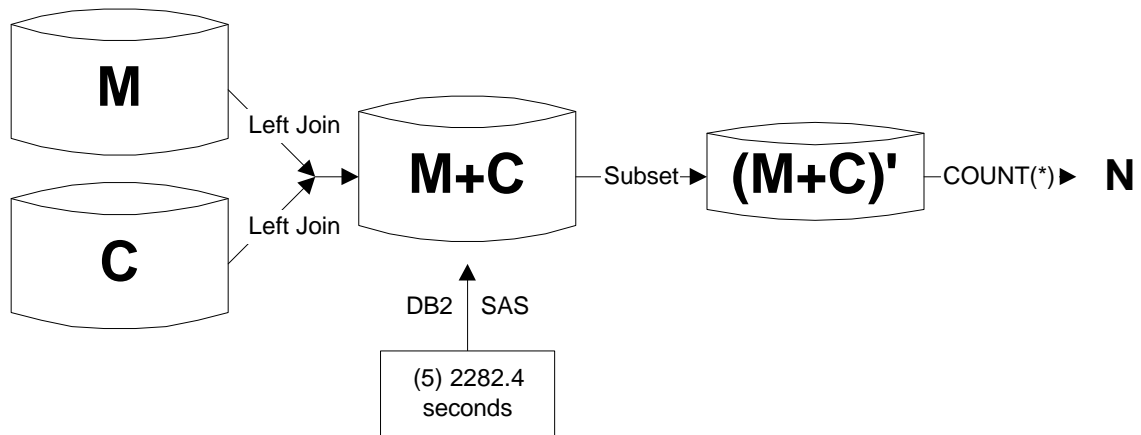
```
SELECT COUNT(*)
FROM   (SELECT *
       FROM   work.m) AS m
LEFT JOIN
      work.c
ON     m.account=c.account
AND   m.time=c.time;
```

(3) Pass-Through SQL:

```
SELECT COUNT(*)
FROM   CONNECTION TO db2
      (SELECT *
       FROM   (SELECT *
              FROM   m
              WHERE  SUBSTR(code,1,1) = 'A') AS m
       LEFT JOIN
            c
       ON     m.account=c.account
       AND   m.time=c.time);
```

(4) Pass-Through SQL:

```
SELECT *
FROM   CONNECTION TO db2
      (SELECT COUNT(*)
       FROM   (SELECT *
              FROM   m
              WHERE  SUBSTR(code,1,1) = 'A') AS m
       LEFT JOIN
            c
       ON     m.account=c.account
       AND   m.time=c.time);
```



Sample code for the above flowchart:

(5) Pass-Through SQL:

```

SELECT COUNT(*)
FROM CONNECTION TO db2
  (SELECT *
   FROM m
   LEFT JOIN
     c
   ON m.account=c.account
      AND m.time=c.time)
WHERE SUBSTR(code,1,1) = 'A';
  
```

3.4. Selecting Rows from a SAS Dataset using another SAS Dataset

This example uses data from 2 SAS datasets with common columns, but there are duplicate column values in both tables. Note that this process can also be performed in DB2. The rows from the 1st SAS dataset will be selected if there is, or is not, a match between a common column. No additional information in the 2nd table is required.

There were 2 SAS datasets used to measure the following results: AX (80,000 rows), and XC (52,000 rows).

3.4.1. Selecting Matching Rows

Sample code for Selecting Matching Rows using the "Where In" Method:

```

SELECT * FROM ax WHERE column IN (SELECT column FROM xc);
  
```

4,300 records produced in 4.1 seconds.

Sample code for Selecting Matching Rows using the "Inner Join" Method:

```

SELECT * FROM ax, xc WHERE ax.column=xc.column;
  
```

10,595,900 records produced in 122.4 seconds.

Sample code for Selecting Matching Rows using the "Distinct Inner Join" Method:

```

SELECT DISTINCT * FROM ax, xc WHERE ax.column=xc.column;
  
```

4,300 records produced in 303.8 seconds.

3.4.2. Selecting Missing Rows

Sample code for Selecting Missing Rows using the "Where Not In" Method:

```

SELECT * FROM ax WHERE column NOT IN (SELECT column FROM xc);
  
```

75,600 records produced in 4.5 seconds.

Sample code for Selecting Missing Rows using the "Distinct Left Join" Method:

```
SELECT * FROM ax LEFT JOIN xc ON ax.column=xc.column
WHERE xc.column=" ";
```

75,600 records produced in 175.4 seconds.

4. Summary of Results

4.1. Summarising Rows in a DB2 Table (3.1)

The time taken to summarise the data decreases when most of the processing is performed in DB2 before transferring the result to SAS.

4.2. Combining DB2 Tables using Inner Join (3.2) or Left Join (3.3)

As above in 3.1 the time taken decreases as more processing is performed in DB2, but also the early subsetting of the data reduces the amount of data being processed at any time.

4.3. Selecting Rows from a SAS Dataset using another SAS Dataset

4.3.1. Selecting Matching Rows (3.4.1)

The presence of duplicate records in both base and look-up SAS datasets creates additional overheads when using the "Inner Join" method, because each duplicate record in one table matches with each duplicate record in the other, and removing these additional matches with the Distinct keyword can more than double the processing time. The "Where In" method only has to scan the look-up SAS dataset each time there is a new base record until the first match is found.

4.3.2. Selecting Missing Rows (3.4.2)

The presence of duplicate records in both base and look-up SAS datasets creates additional overheads when using the "Left Join" method, as in 3.4.1. The "Where Not In" method has to scan the full look-up SAS dataset each time there is a new base record, if no match is found.

5. Recommendations

- The "Left Join" method is now available in DB2 version 4, which removes the necessity to perform separate "Where Not In" and "Inner Join" processes in DB2, then combining the fragments in SAS.
- If duplicate values exist in the column used for matching, then avoid joining the tables. If no additional information is needed from one of the tables, then use "Where In" or "Where Not In" processing to just select the records required.
- Access Descriptors are designed for simple updating of small DB2 tables using SAS/FSP® or SAS/AF® applications and utilities. Do not use Access Descriptors for reading large DB2 tables.
- If the DB2 tables are indexed, try to make use of the indexed columns for joining, and carry out as much processing within DB2 before passing the results to SAS. Even if the SAS processing is faster than DB2, the time required to copy a large amount of data from DB2 to SAS can be prohibitive. In other words, reduce the data as much as possible before moving it into SAS.

6. References and further reading

- SAS/ACCESS® Software for Relational Databases: Reference, Version 6, First Edition
- SAS® Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition

The author is a consultant for Holland Numerics Ltd and can be contacted at the following address:

Philip R Holland
Holland Numerics Ltd
94 Green Drift
Royston
Herts. SG8 5BT
UK
e-mail: <phil.holland@bcs.org.uk>
tel. (mobile): +44-(0)7714-279085

SAS, SAS/AF, SAS/FSP and SAS/ACCESS are a registered trademarks of SAS Institute Inc., Cary, NC, USA.