Paper 275-27

SAS Accessing DB2 Data: a Performance Victory Away

ir. Henri Theuwissen, SOLID Partners, Belgium
Nancy Croonen, CC Training Services, Belgium

ABSTRACT

Many MVS applications require data that is stored in DB2 tables. In the past a common statement was: “Do as much as possible, as close as possible to the source data to get the best performance” or “execute as much as possible in DB2”.
This paper covers the exploitation of DB2 data. Based on real life projects, tips and guidelines are discussed on how to get the best performance in your SAS job. Tests on production environments often give surprising results:

- Combining DB2 data can be faster in SAS than combining it in DB2.
- Consolidating data in SAS, using the SUMMARY procedure is often much faster than using the SQL GROUP BY clause in DB2.

When DB2 data must be combined with non-DB2 data it is even not possible to execute the jobs in DB2.

INTRODUCTION

The paper contains four parts:

1. The various possibilities to access DB2 data. Advantages and disadvantages for each method are given.
2. New features to improve the performance in DB2 access that were introduced with Version 8.
3. Different possibilities to combine data from DB2 tables only.
4. Different methods to combine DB2 data with SAS data.

The examples presented in this paper come from various projects at different customer sites. All the sites run SAS Version 8 on MVS.
The tests were executed using several SAS system options to get information about the SAS processing.

```
OPTIONS SASTRACELOC=SASLOG FULLSTIMER
        SASTRACE=',,,d';
```

THE BUSINESS CASES

A SHARES AND FUNDS DATA MART

A company sells shares and funds. Information about each share or fund is stored in a DB2 table called FUNDLOOKUP with less than 1.000 rows. Daily buy and sale transactions are stored in a fact table called FUNDFACTS with more than 300.000 rows.
The common column between the two tables is the fund identifier. There is a 1-to-many relationship between the two tables: for each share or fund in the FUNDLOOKUP table there can be one or more rows in the FUNDFACTS table.
For each row in the FUNDFACTS table, a corresponding row with the same fund identifier must exist in the FUNDLOOKUP table.

A HUMAN RESOURCE MANAGEMENT SYSTEM

Human Resource data are stored in a DB2 database. Several operational reports must be created from this system. Data are stored in a normalized schema.
Each employee has an employment contract with one employer. Data about the employer are stored in the COMPANY table. The fixed, basic data about the employment contract are stored in the table CONTRACT (e.g. entrance date in the company, contract number, …). Each employee may have several detail records, for which historical information is stored; for example an employee started on a full time basis and changed afterwards to part time employment. This information is stored in the CONTRHIST table. For each employee, his/her activities (working, holiday, sick, …) are stored on a daily basis in the table ACTIVITIES.

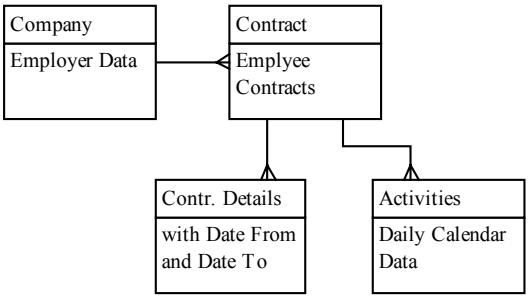


Fig. 1: Partial Table Structure of the HR database

The employee number is a common column for the three tables CONTRACT, CONTRHIST and ACTIVITIES. The company number is the common column for the table CONTRACT and the table COMPANY.
There is a 1-to-many relationship between COMPANY and CONTRACT: each company has one or more employees. There is a 1-to-many relationship between CONTRACT and CONTRHIST: each employee has one or more historical records. Finally there is a 1-to-many relationship between CONTRACT and ACTIVITIES: each employee can have one or more activity rows.

	Company	Contracts	Contract Details	Daily Activities
Rows	50 K	650 K	2.000 K	250.000 K

Table 1: Table size (number of rows)

ACCESSING DB2 DATA

SAS software provides three methods to get transparent access to DB2 data:

1. Access and view descriptors
2. The SQL Passthru facility
3. SAS/ACCESS Libname engine

ACCESS AND VIEW DESCRIPTORS

Access and view descriptors were introduced in Version 6.06 of SAS software. The main reason for introducing access descriptors was to circumvent the limit of 8 characters for SAS variable names when accessing long DBMS column names.

New features for access descriptors in Version 8 include update possibilities: the UPDATE statement in the ACCESS procedure identifies an existing access or view descriptor that you want to change.

Disadvantages of the access and view descriptors include:

- An additional maintenance layer is added for the DB2 DBA, since table and column definitions must be maintained in DB2 and in SAS.
- Long DB2 column names are truncated to a maximum of 8 positions.
- Combining two DB2 tables in SAS, through access and view descriptors is very inefficient because both tables are transferred first to SAS and then combined in SAS.

Guidelines for view descriptors indicate to specify only these columns that are required in the SAS job. Even if your query only requests a few columns, all columns specified in the view descriptor are read into SAS. When processing large tables, there is only a very small gain in performance, as shown in the following example: a SAS job requires 2 columns from a large DB2 table. The result is created using a view descriptor with only 2 columns and with a view descriptor containing all columns.

```
PROC ACCESS DBMS=DB2;
   CREATE WORK.CONTRACT.ACCESS;
   SSID=DB2P;
   TABLE DB2P.CONTRACT;
   ASSIGN=YES;

   CREATE WORK.CONTFULL.VIEW;
   SELECT ALL;

   CREATE WORK.CONTLMTD.VIEW;
   SELECT CON_EMPLOYEE_NR
          CON_COMPANY_NR;
RUN;

PROC SQL;
   CREATE TABLE FULLVIEW AS
   SELECT CON_EMPL,
          CON_COMP
   FROM CONTFULL;
QUIT;

PROC SQL;
   CREATE TABLE LMTDVIEW AS
   SELECT CON_EMPL,
          CON_COMP
   FROM CONTLMTD;
QUIT;
```

	All Columns in View Descriptor	Few Columns in View Descriptor
CPU Time (seconds)	56.96	56.76

Table 2: CPU Time comparison view descriptors

SQL PASSTHRU FACILITY

The SQL Procedure Passthru facility uses SAS/ACCESS software to connect to DB2 and to send statements directly to DB2 for execution. It allows using the SQL syntax of DB2, and it supports any non-ANSI standard SQL that is supported by DB2.

DB2 and the SAS System have different naming conventions. Prior to Version 8, SAS column names were still limited to a maximum of 8 characters whereas DB2 allowed longer column names. Therefore some DB2 column names were changed when you retrieved DB2 data through the CONNECTION TO component. Beginning with Version 8, SAS column names can be up to 32 characters long. So these longer SAS column names also improve compatibility with DB2 data that allow longer column names.
This enhancement is illustrated with the following example: the company table COMPANY in the HR database contains 25 columns. A few of them are listed below:

- CPY_NAME_PART1
- CPY_NAME_PART2
- CPY_NR
- CPY_ADRESS
- CPY_POSTAL_CODE

A SAS copy of this DB2 table with data for one location is created using the SQL Passthru facility, by executing these statements:

```
PROC SQL;
   CREATE TABLE LEUVEN AS
   SELECT * FROM CONNECTION TO DB2
      (SELECT *
       FROM  DB2P.COMPANY CPY
       WHERE CPY.CPY_POSTAL_CODE=3000);
```

This query is executed using version 6.09E and 8.2. A partial structure of the SAS table LEUVEN is shown in Table 3.

Version 6.09E	Version 8.2
CPY_NAME	CPY_NAME_PART1
CPY_NAM0	CPY_NAME_PART2
CPY_NR	CPY_NR
CPY_ADRE	CPY_ADRESS1
CPY_POST	CPY_POSTAL_CODE

Table 3: Table structure from SQL Passthru

An advantage of the SQL Passthru facility compared to access and view descriptors is that no additional maintenance is required. Additionally many QMF queries might exist already and they can be used without any change in SAS.
Disadvantages of the SQL Passthru facility to access DB2 data include:

- The query that is passed to DB2 must follow the DB2 SQL syntax. There are a lot of small differences, compared to the SAS SQL syntax. For example, within DB2 comparison operators must be specified as >, <, >=, etc. DB2 does not allow the usage of GT, LT, GE etc.

- The SQL passed to DB2 is limited to the manipulations and functions that are supported by DB2. Many powerful SAS functions are not allowed, unlike SQL in SAS.

THE SAS/ACCESS LIBNAME ENGINE

Starting from Version 8, SAS allows direct access to DB2 tables through the SAS/ACCESS Libname statement.

The general syntax to access MVS DB2 data is shown below:

```
LIBNAME LIBREF DB2
                AUTHID=DB2_AUTHORIZATION_ID
                OTHER_DB2_ENGINE_OPTIONS
                OTHER_LIBNAME_OPTIONS;
```

Notice that you need read access authorization on the DB2 system table SYSIBM.SYSTABLES to execute a PROC DATASETS on that *libref* or to open the DIR window for the *libref*. Otherwise an error message is shown:

```
ERROR: DB2 prepare error:
       DSNT408I SQLCODE = -551,
ERROR: userid DOES NOT HAVE THE PRIVILEGE TO
       PERFORM OPERATION SELECT ON OBJECT
       SYSIBM.SYSTABLES.
```

With the SAS/ACCESS Libname engine you get full transparency to access data: whether the data is stored in SAS tables or in DB2 tables, no changes to the program statements are required.

PERFORMANCE FEATURES IN VERSION 8

DBKEY= OPTION

The DBKEY= data set option is used to improve performance when joining a small SAS table with a large DB2 table. Notice though that you might get the opposite effect (degrading performance) when this option is not used carefully.
The DBKEY= data set option will generate for each value of the key column a separate query to DB2 returning a result set with 0, 1 or many rows.

The following example illustrates the usage of the DBKEY= option in the HR system from Fig. 2: consider a small SAS table, named SMALL_SET with 4 rows and 1 column, named EMPLOYEE_NR. You need to find the corresponding rows in the DB2 table CONTRACT, which contains over 650.000 rows. The advantage of using the DBKEY= option is compared with the more traditional methods.

A first solution is created, using SQL JOIN operations with the SAS/ACCESS Libname engine.

```
LIBNAME DB DB2 AUTHID=DB2P;

PROC SQL;
   CREATE TABLE TEST AS
   SELECT L.*
   FROM SMALL_SET   S,
        DB.CONTRACT L
   WHERE S.EMPLOYEE_NR=
         L.CON_EMPLOYEE_NR;
QUIT;
```

A second solution is created, using the DBKEY= option.

```
LIBNAME DB DB2 AUTHID=DB2P;

DATA TEST;
   SET SMALL_SET;
   SET DB.CONTRACT (DBKEY=CON_EMPLOYEE_NR)
                   KEY=DBKEY;
RUN;
```

	SQL JOIN	DBKEY option
CPY Time	21.10	0.03
EXCP	1476	5

Table 4: CPU Time DBKEY= option

Using the DBKEY= option can decrease performance when the SAS table is too large. This is because each value in the SAS table generates a new query and result set from the DB2 table. To illustrate this, the previous example was executed again, with 500 rows in the SAS table.

Size SAS table	SQL JOIN	DBKEY option
4 rows	21.10	0.03
500 rows	29.21	91.14

Table 5: DBKEY= option on large SAS tables

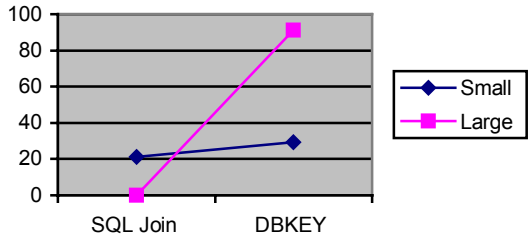


Fig 2.: DBKEY= option on large SAS tables

Be careful when using the DBKEY= option in a DATA step with two SET statements: each DATA step loop will read 1 row from the first table and read one corresponding row from the second table. However, this means that this feature is useful for tables with unique keys. You might get a wrong result if the second table contains multiple rows for the same key value of the first table.
This is illustrated by the following example: the SAS table TEST contains one row, with one column named ACT_EMPLOYEE_NR with a value of 8000000. The DB2 table DB2P.ACTIVITIES contains over 1.000 rows with the value 80000000 for the column ACT_EMPLOYEE_NR. The execution of the following SAS step results in the SAS table WORK.COMBINE with only one row:

```
LIBNAME DB DB2 AUTHID=DB2P;

DATA COMBINE;
   SET TEST;
   SET DB.ACTIVITIES (DBKEY=ACT_EMPLOYEE_NR)
                     KEY=DBKEY;
RUN;
```

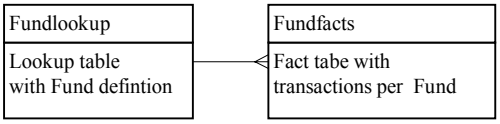
THE IMPLICIT SQL PASSTHRU

Starting from Version 8.2 several SQL-based query optimizations are implemented in accessing underlying DB2 data.
Implicit SQL Passhru converts a PROC SQL query into a SQL query that is specific for the underlying DBMS.

UNION operation

UNION operations are passed to DB2. Since a UNION operation eliminates duplicate rows, the processing will be more efficient when passed to DB2.
This feature is illustrated below on the Shares and Funds Data Mart: you need to find the unique fund identification numbers from the lookup file FUNDLOOKUP and the details file FUNDFACTS. (This query is created for demo purpose only since, according to the database structure all fund identification numbers from the table FUNDFACTS must also appear in the table FUNDLOOKUP).

Fig. 3: Partial Structure Share and Funds Data Mart



The result is created in three different approaches using the SQL procedure.

First a traditional SQL Passthru facility is used:

```
PROC SQL;
   SELECT * FROM CONNECTION TO DB2
     (SELECT FUND_ID FROM DB2P.FUNDLOOKUP
      UNION
      SELECT FUND_ID FROM DB2P.FUNDFACTS);
```

Notes in the LOG window indicate:

```
Prepare stmt: select FUND_ID from
      DB2P.FUNDLOOKUP union SELECT FUND_ID from
      DB2P.FUNDFACTS
```

Then the same result is created using a UNION in SAS, after executing two separate SQL Passthru steps:

```
PROC SQL;
   SELECT * FROM CONNECTION TO DB2
      (SELECT FUND_ID FROM DB2P.FUNDLOOKUP)
   UNION
   SELECT * FROM CONNECTION TO DB2
      (SELECT FUND_ID FROM DB2P.FUNDFACTS);
```

Notes in the LOG window show that two separate queries are executed on DB2:

```
Prepare stmt:select FUND_ID from DB2P.FUNDLOOKUP

Prepare stmt:select FUND_ID from DB2P.FUNDFACTS
```

Finally the UNION is created using the SAS/ACCESS Libname engine:

```
LIBNAME DB DB2 AUTHID=DB2P;

PROC SQL;
   SELECT FUND_ID FROM DB.FUNDLOOKUP
   UNION
   SELECT FUND_ID FROM DB.FUNDFACTS;
```

The prepare statement message in the LOG window clearly proves that the query is converted into a DB2 query and passed to DB2:

```
SQL Implicit Passthru stmt prepared is:
  select  DB2P.FUNDLOOKUP."FUND_ID" from
  DB2P.FUNDLOOKUP union select
  DB2P.FUNDFACTS."FUND_ID" from DB2P.FUNDFACTS
```

	SQL Passthru 1 step	SQL Passthru 2 steps	SAS/ACCESS LIBNAME engine
CPU time	0.02	4.30	1.58
EXCP	32	592	198

Table 6: CPU Time Implicit SQL Passthru with UNION

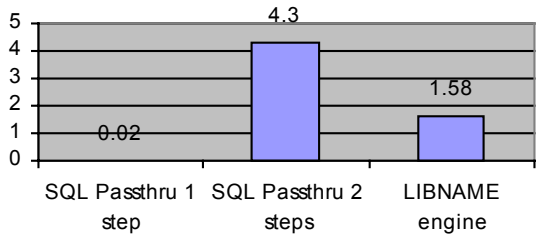


Fig. 4: CPU Time Implicit SQL Passthru with UNION

JOIN operation

Suppose you need to combine the lookup table FUNDLOOKUP and the facts table FUNDFACTS from the Shares and Funds Data Mart. This request is solved, using three different methods:

1. DATA step processing with the SAS/ACCESS Libname engine.
2. SQL JOIN with the SAS/ACCESS Libname engine.
3. SQL Passhru facility.

Within the DATA step MERGE, the data is extracted from DB2, and combined in SAS.

```
LIBNAME DB DB2 AUTHID=DB2P;

DATA COMBINE;
   MERGE DB.FUNDLOOKUP (IN=A)
         DB.FUNDFACTS  (IN=B);
   BY FUND_ID;
   IF A AND B;
```

When using the SQL JOIN with the SAS/ACCESS Libname engine, the join query is passed to DB2 and executed in DB2. This results in fewer rows to be transferred to SAS, compared to the join in SAS, where both tables are first completely transferred to SAS.

```
LIBNAME DB DB2 AUTHID=DB2P;

PROC SQL;
   CREATE TABLE COMBINE AS
   SELECT *
```

```
      FROM  DB.FUNDLOOKUP L
            DB.FUNDFACTS  F
      WHERE L.FUND_ID=F.FUND_ID;
```

Within the LOG window a message is printed, indicating that SQL Implicit Passthru is used.

```
SQL Implicit Passthru stmt prepared is:
  select DB2P.FUNDLOOKUP."FUND_ID",
         DB2P.FUNDLOOKUP."FUND_DESCRIPTION",
         …
         DB2P.FUNDFACTS."AMOUNT"
  from DB2P.FUNDLOOKUP, DB2P.FUNDFACTS where
```

Finally the SQL Passthru facility is used.

```
   PROC SQL;
      CREATE TABLE COMBINE AS
      SELECT * FROM CONNECTION TO DB2 (
         SELECT *
         FROM  DB2P.FUNDLOOKUP L,
               DB2P.FUNDFACTS  F
         WHERE L.FUND_ID=F.FUND_ID);
   QUIT;
```

	Data Step	Implicit Passhru	SQL Passthru
CPU time	46.02	39.43	17.99

Table 7: CPU Time Implicit SQL Passthru with JOIN

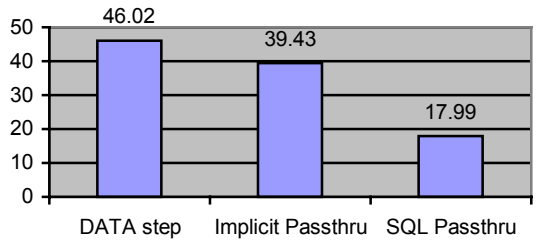


Fig. 5: CPU Time Implicit SQL Passthru with JOIN

WHERE CLAUSE OPTIMIZATIONS

The WHERE clause optimizer passes SAS WHERE clauses to the underlying DB2. This facility is implemented in SAS/ACCESS since Version 6. Beginning in Version 8.2, many SAS functions get also passed to the DB2. If DB2 has an equivalent function, the SAS function gets passed down. Examples of these functions include LOWCASE, UPCASE, SUBSTR.

The following example illustrates the WHERE clause optimizer, as it has been available since Version 6. An access and view descriptor are defined on the table CONTRACTS in the HR database. The view descriptor is called WORK.CONTRACTS.

```
DATA _NULL_;
   SET CONTRACTS;
   IF CON_COMPANY_NR = 800000;
RUN;

DATA _NULL_;
   SET CONTRACTS;
   WHERE CON_COMPANY_NR = 800000;
RUN;
```

```
DATA _NULL_;
   SET CONTRACTS;
   WHERE YEAR(CON_START_DT) = 2001;
RUN;
```

The first DATA step reads data from the HR system, using a subsetting IF statement. All DB2 data is read in SAS, and the subset is created in SAS.
In the second DATA step, the WHERE clause is passed to DB2, and the subset is created in DB2. Only the result is transferred to SAS.
In the third DATA step, a SAS function is used in the WHERE clause, and therefore all data is transferred to SAS, and the subset is created over there.

	IF	WHERE	WHERE with SAS function
CPU time	16.28	0.02	17.76

Table 8: CPU Time WHERE clause optimization in Version 6

A second example examines the behavior in Version 8.2. Several queries are executed to get data from the DB2 table FUNDFACTS from the Shares and Funds Data Mart.
The goal of this example is to test which WHERE clause gets passed to DB2.

```
LIBNAME DB DB2 AUTHID=DB2P;

PROC SQL ;
   CREATE TABLE SUBSET AS
      SELECT *
      FROM DB.FUNDFACTS F
      WHERE F.OFFICE='XYZ';
QUIT;

PROC SQL ;
   CREATE TABLE SUBSET AS
      SELECT *
      FROM DB.FUNDFACTS F
      WHERE LOWCASE(F.OFFICE)='xyz';
QUIT;

PROC SQL ;
   CREATE TABLE SUBSET AS
      SELECT *
      FROM DB.FUNDFACTS F
      WHERE F.OFFICE !! F.LOCATION='XYZ3000';
QUIT;

PROC SQL ;
   CREATE TABLE SUBSET AS
      SELECT *
      FROM DB.FUNDFACTS F
      WHERE SCAN(F.OFFICE,1, ' ')='XYZ';
QUIT;
```

The first SQL query passes the WHERE clause to DB2, since it only contains the equals to operator.
Within the second and third SQL query, functions are used (CONCATENATION, LOWCASE) that have an equivalent in DB2 and therefore the WHERE clause is passed to DB2.
In the last QUERY a SAS function (SCAN) is used, with no equivalent function in DB2. As a result the WHERE clause is executed in SAS.

	Equals to	Concatenation	Lowcase	Scan
CPU Time	7.05	7.12	7.36	10.26

Table 9: CPU Time WHERE clause optimization

The following example illustrates the performance optimization when using the SAS/ACCESS Libname engine and WHERE clauses: you need to combine the data from the tables FUNDLOOKUP and FUNDFACTS from the Share and Funds Data Mart. You are only interested in two fund identification values: 40 and 270. (Remember that the FUNDLOOKUP table contains approx. 1.000 rows and the FUNDFACTS table contains more than 300.000 rows).

Combine these two tables, using a WHERE clause on the FUND_ID.

```
LIBNAME DB DB2 AUTHID=DB2P;

DATA COMBINE;
   MERGE DB.FUNDLOOKUP
         DB.FUNDFACTS;
   BY FUND_ID;
   WHERE FUND_ID IN (40, 270);
QUIT;
```

The LOG window shows the following messages:

```
NOTE: There were 2 observations read from the
      data set DB2P.FUNDLOOKUP.
      WHERE FUND_ID IN (40, 270);
NOTE: There were 53 observations read from
      the data set DB2P.FUNDFACTS.
      WHERE FUND_ID IN (40, 270);
NOTE: The DATA statement used the following
      resources:
        CPU     time -         00:00:46.02
```

Combine these two tables, using an IF statement on the FUND_ID.

```
LIBNAME DB DB2 AUTHID=DB2P;

DATA COMBINE;
   MERGE DB.FUNDLOOKUP
         DB.FUNDFACTS;
   BY FUND_ID;
   IF FUND_ID=40 OR FUND_ID=270;
RUN;
```

The LOG window shows the following messages:

```
NOTE: There were 819 observations read from
      the data set DB2P.FUNDLOOKUP.
NOTE: There were 305569 observations read
      from the data set DB2P.FUNDFACTS.
NOTE: The DATA statement used the following
      resources:
      CPU     time -         00:00:46.02
```

COMBINING DB2 DATA

The topics described below provide a comparison of exploiting data stored in DB2 tables. No information is residing in other DBMS stores or in SAS tables.

EXTRACTING DETAILED INFORMATION FROM DB2

To compare different approaches, a test is executed on the HR database. The goal of the application is to provide information about the working activities of employees of a specific company. This requires data from the four DB2 tables in Fig.2.

The result is created using the SQL Passthrough facility. Three different solutions are compared to get the result:

1. Using an INNER JOIN of the four tables
2. Using a subquery
3. Split the job in two separate tasks, using macro variables

Using the INNER JOIN on the four tables is the easiest way of programming.

```
PROC SQL;
   CREATE TABLE EMP AS
   SELECT * FROM CONNECTION TO DB2 (
      SELECT COLUMNS
      FROM DB2P.COMPANY,
           DB2P.CONTRACT,
           DB2P.CONTRHIST,
           DB2P.ACTIVITIES
      WHERE COMBINE_CONDITIONS);
```

With the subquery, the large ACTIVITIES table is queried with a WHERE clause on ACT_EMPLOYEE_NR. The value must exist in the result of a subquery on the combination of the other three tables.

```
PROC SQL;
   CREATE TABLE EMP AS
   SELECT * FROM CONNECTION TO DB2 (
      SELECT COLUMNS
      FROM DB2P.ACTIVITIES
      WHERE ACT_EMPLYEE_NR IN
          (SELECT CON_EMPLOYEE_NR
           FROM DB2P.COMPANY,
                DB2P.CONTRACT,
                DB2P.CONTRHIST
           WHERE COMBINE_CONDITIONS));
```

The last method will first search for the required employees in the combination of three tables COMPANY, CONTRACT and CONTRHIST. This gives a set of employee numbers. A macro variable is created containing the employee numbers and this macro variable is used to build a WHERE clause used in a second query on the ACTIVITIES table.

```
PROC SQL;
   CREATE TABLE EMP1 AS
   SELECT * FROM CONNECTION TO DB2 (
      SELECT COLUMNS
      FROM DB2P.COMPANY,
           DB2P.CONTRACT,
           DB2P.CONTRHIST
      WHERE COMBINE_CONDITIONS);
```

```
QUIT;

PROC SQL NOPRINT;
   SELECT DISTINCT CON_EMPLOYEE_NR
          INTO :EMPNRS
          SEPARATED BY ','
   FROM EMP1;
QUIT;


PROC SQL;
   CREATE TABLE EMP2 AS
   SELECT * FROM CONNECTION TO DB2 (
      SELECT COLUMNS
      FROM DB2P.ACTIVITIES
      WHERE ACT_EMPLOYEE_NR IN
            ( &EMPNRS)  );
QUIT;

PROC SQL;
   SELECT *
   FROM EMP1,
        EMP2
   WHERE EMP1.CON_EMPLOYEE_NR=
         EMP2.ACT_EMPLOYEE_NR;
QUIT;
```

This method will provide an excellent performance. The only limitation is that the total length of the concatenation of all key values to be stored in the macro variable may not exceed 32K.

	FULL JOIN	SUBQUERY	SQL INTO
CPU Time	361.27	350.16	1.53

Table 10: CPU Time combining DB2 data

SUMMARIZING DATA

Data consolidation can be executed in DB2, close to the source data or in SAS. Intuitively you might believe that summarizing the data in DB2 is faster, since only a small consolidated subset must be transferred to SAS. However, in some cases we encountered the opposite.

Suppose that you need to create a report on the HR database, showing for each employee the total hours worked. The report must show total values by the combination company number – employee number.

The first method uses the SUM function in DB2, combined with a GROUP BY statement.

```
PROC SQL;
   CREATE TABLE HOLIDAY AS
   SELECT * FROM CONNECTION TO DB2 (
      SELECT COLUMNS,
             SUM(ACT_HOURS) AS HOLS
      FROM DB2P.COMPANY,
           DB2P.CONTRACT,
           DB2P.CONTRHIST,
           DB2P.ACTIVITIES
      WHERE COMBINE_CONDITIONS
      GROUP BY CON_COMPANY_NR,
               CON_EMPLOYEE_NR);
QUIT;
```

The result contains approx. 14.000 observations.

The second method extracts the detailed data into a SAS table, followed by a SUMMARY procedure on this table.

```
PROC SQL;
   CREATE TABLE HOL_DETAIL AS
   SELECT * FROM CONNECTION TO DB2 (
      SELECT COLUMNS
      FROM DB2P.COMPANY,
           DB2P.CONTRACT,
           DB2P.CONTRHIST,
           DB2P.ACTIVITIES
      WHERE COMBINE_CONDITIONS);
QUIT;

PROC SUMMARY DATA=HOL_DETAIL NWAY;
   CLASS CON_COMPANY_NR
         CON_EMPLOYEE_NR;
   VAR ACT_HOURS;
   OUTPUT OUT=HOLIDAY SUM=;
RUN;
```

	GROUP BY in DB2	SUMMARY in SAS
SQL step	61.96	42.30
Summary Procedure	N/A	1.26
Total	61.96	43.56

Table 11: CPU Time summarizing DB2 data

Within the second method, more data is extracted from DB2, written to a SAS table and read again for the SUMMARY procedure. This results in more I/O and a higher EXCP count. Still the CPU time decreased with over 30 %.

COMBINING SAS DATA AND DB2 DATA

Many applications require a combination of SAS data and DB2 data. The combination must happen either in SAS or in DB2. This implies that either the SAS data must be transferred into DB2, or the DB2 table must be extracted into SAS. To transfer the SAS data into a temporary DB2 table or DB2 view requires TABLE CREATE authorizations in the DB2 environment. Very often you are not granted this functionality.

Consider the DB2 table FUNDFACTS containing detailed information about buy ans sale transactions. A SAS table, called SAS_TABLE contains 500 rows and 1 column. Each row contains the identification number of one fund or share. You need to get all data from the DB2 table for the funds or shares in the SAS table. Several approaches are examined:

1. DATA step MERGE using the SAS/ACCESS Libname engine.
2. SQL JOIN using the SAS/ACCESS Libname engine.
3. DATA step SET – SET operation, using the DBKEY= option and the SAS/ACCESS Libname engine.
4. Creation of a macro variable on the small SAS data set and using this macro variable in a SQL WHERE clause with the SAS/ACCESS Libname engine.
5. Creation of a macro variable on the small SAS data set and using this macro variable in a SQL WHERE clause with the SQL Passthru facility.

These five methods are examined below.

First, using the SAS/ACCESS Libname engine, a DATA step MERGE is executed. The full DB2 table is read into SAS and the data is combined in SAS.

```
LIBNAME DB DB2 AUTHID=DB2P;

DATA COMBINE;
   MERGE SAS_TABLE (IN=A)
         DB.FUNDFACTS;
   BY FUND_ID;
   IF A;
RUN;
```

Secondly, using the SAS/ACCESS Libname engine, an SQL JOIN is executed.

```
LIBNAME DB DB2 AUTHID=DB2P;

PROC SQL;
   CREATE TABLE COMBINE AS
   SELECT *
   FROM SAS_TABLE,
        DB.FUNDFACTS
   WHERE SAS_TABLE.FUND_ID =
         FUNDFACTS.FUND_ID;
QUIT;
```

As a third method, a DATA step is executed with two SET statements. The second SET statement specifies the DBKEY= data set option and uses the SAS/ACCESS Libname engine. Since the SAS table is (too) large, you do not benefit from the DBKEY= data set option. On the contrary, there is a performance degrade. The DBNULLKEYS=YES or DBNULLKEYS=NO data set option has no effect on the performance in this example.

```
LIBNAME DB DB2 AUTHID=DB2P;

DATA COMBINE;
   SET SAS_TABLE;
   SET DB.FUNDFACTS(DBKEY=FUND_ID)
       KEY=DBKEY;
RUN;
```

As a fourth approach, a macro variable is created, containing the distinct key values of the SAS table. This macro variable is used in a WHERE clause used in an SQL query using the SAS/ACCESS Libname engine. In this case, no data joining process is required. You just run a simple query on 1 table, with a simple WHERE clause. Notice that the total length of your macro variable is limited to 32K.

```
LIBNAME DB DB2 AUTHID=DB2P;

PROC SQL NOPRINT;
   SELECT DISTINCT FUND_ID
          INTO :NRS SEPARATED BY ', '
   FROM SAS_TABLE;
QUIT;

PROC SQL;
   CREATE TABLE COMBINE AS
   SELECT *
   FROM DB.FUNDFACTS
   WHERE FUND_ID IN ( &NRS );
QUIT;
```

As a final solution, a macro variable is created, containing the distinct key values of the SAS table. This macro variable is used in a WHERE clause used in the SQL Passthru facility. The same remark as in the previous example is valid for this one.

```
LIBNAME DB DB2 AUTHID=DB2P;

PROC SQL NOPRINT;
   SELECT DISTINCT FUND_ID
          INTO :NRS SEPARATED BY ', '
   FROM SAS_TABLE;
QUIT;

PROC SQL;
   CREATE TABLE COMBINE AS
   SELECT * FROM CONNECTION TO DB2 (
     SELECT *
     FROM DB2P.FUNDFACTS
     WHERE FUND_ID IN ( &NRS )  );
QUIT;
```

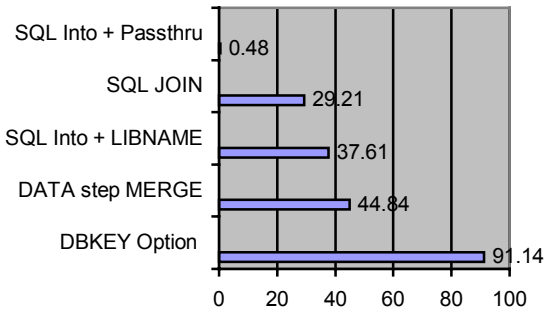


Fig. 6: CPU Time combining SAS data with DB2 data

CONCLUSION

Evolving from version 6.06 to 8.2, SAS has offered various methods to access DB2 data. Each of the available methods has advantages and disadvantages. Depending on your applications you have to choose between performance and ease of use: the SAS/ACCESS Libname engine is the most convenient to use, but for performance reasons you might need to use the SQL Passthru facility.

With the development and deployment of Version 8.2, SAS has implemented several optimizations in SQL to improve the performance of DB2 access. Some of them (like the DBKEY= option) must be used carefully.

When combining DB2 data, it can be useful to analyze the requirements, and instead of using SQL queries directly in DB2, you might split the task in several queries and use the full power of SAS data manipulation capabilities. Especially the creation of macro variables, passed into SQL WHERE clauses is recommended.

If you need to combine SAS tables with DB2 tables, the 'best' method depends on the size of the tables that you want to combine: for small tables you might use the SAS/ACCESS Libname engine with the DBKEY= option; for medium size SAS tables the best performance is given by SQL Passthru, in combination with macro variables to create the WHERE clause.

The more knowledge you have about the structure and the size of the data that you need to treat, the better you can make wise decisions about which method will give the best results.

REFERENCES

Fred Levine (2001) “Using the SAS/ACCESS Libname Technology to Get Improvements in Performance and Optimizations in SAS/SQL Queries”. Proceedings of the twenty-sixth Annual SAS Users Group International Conference, 26.

Croonen Nancy (2001) “New Features in Version 8 of SAS software”. Course Notes.

Theuwissen Henri (2001) “Efficiency Techniques in SAS Jobs”.

TRADEMARKS

CONTACT INFORMATION

Contact the authors at:

ir. Henri Theuwissen
SOLID Partners NV
Minervastraat 14 Bis
B-1930 Zaventem
Belgium
Work Phone: +32 495 54 52 53
Fax: +32 2 706 03 09
Email: henri.theuwissen@solidpartners.be
Web: www.solidpartners.be

Nancy Croonen
CC Training Services BVBA
Kesseldallaan 12 / 202
B-3010 Kessel-Lo
Belgium
Work Phone: +32 496 28 45 28
Fax: +32 2 706 03 09
Email: nancy.croonen@solidpartners.be
Web: www.solidpartners.be