SQL News You Can Use

March, 1988

Nancy Wheeler

IBM
APL Development
General Products Division
Santa Teresa Laboratory
San Jose, California

Abstract

What do the new features announced recently in IBM Database 2 (DB2), SQL/Data System (SQL/DS) and APL2 mean to you? How can you use these features to the best advantage? This paper contains the answers to those questions, and some hints for how to handle common problems using both the new features and some old features you may not have been aware of.

Contents

Figures

Announcements

IBM Database 2

The newest release of DB2 is Version 1 Release 3. This release adds support for 4 new datatypes (DATE, TIME, TIMESTAMP and REAL), as well as some SQL language enhancements and performance improvements.

The DATE, TIME, and TIMESTAMP datatypes provide the ability to designate a column as a date, time or timestamp. There are many new facilities built into SQL to allow manipulation of these columns, including sorting, arithmetic, and automatic insertion of the current date or time. The data may be formatted in the standard forms for dates and times (ISO, USA, EUR and JIS).

The REAL datatype provides the capability to store floating point numbers in single-precision form. In prior releases of DB2, floating point numbers could be stored only in double-precision form (the FLOAT datatype). The additional option of using single-precision floating point can be a significant space saver where the accuracy of double precision is not required.

The SQL language enhancements include new scalar functions such as LENGTH, SUBSTR and VALUE. Some restrictions on the UNION and LIKE operations have been lifted, and arithmetic expression errors no longer cause processing to stop for the entire table. Instead, a warning message is issued and processing continues.

Details on how to use the new datatypes are contained in this paper. The other new features will be used by APL2 automatically if coded into the APL2 program's SQL statements. For more information on those features, see the appropriate DB2 publications.

SQL/Data System

The newest release of SQL/DS is Version 2 Release 1. This release adds support for the DATE, TIME, and TIMESTAMP datatypes, allows VARCHAR columns larger than 254 characters wide, provides programmed database switching, and also contains other miscellaneous improvements.

The DATE, TIME and TIMESTAMP implementation is compatible with the DB2 implementation described in the previous section.

The VARCHAR extension provides compatibility with DB2, and allows programs to allocate storage more efficiently.

Database switching is provided in SQL/DS Version 2 with extensions to the CONNECT command which allow the specification of a database name in addition to the user id and password. Use of these extensions requires the TSAF facility of VM/SP5.

More details about using the new datatypes and VARCHAR and CONNECT extensions with APL are provided in this paper. The other new features will be used by APL2 automatically at execution time where applicable. For more information about those features, see the appropriate SQL/DS documentation.

APL2

The newest release of APL2 is Version 1 Release 3. The following enhancements to AP 127, the APL2 SQL interface, are provided in APL2 Version 1 Release 3:

- Support for the new datatypes provided in DB2 and SQL/DS
- Support for use of DBCS data
- Programmable Isolation Level switching
- Additional information with DESCRIBE command
- Prototypes for Empty Results

More detail about these new features will be provided in the rest of this paper.

Usage of New Features

Datetime Datatypes

The DATE, TIME and TIMESTAMP datatypes are chosen by coding those types in the SQL CREATE TABLE statement.

Data is passed to SQL for these columns in character string form, just as for a CHAR or VARCHAR column. When the column has been declared as DATE, TIME, or TIMESTAMP, SQL will interpret the character string, make sure it is a valid date or time, and store it in a special internal format. This internal format allows SQL to manipulate the dates to provide the built-in sorting and arithmetic capabilities.

When installing SQL/DS or DB2, one of the four standard formats for dates and times (ISO, USA, EUR or JIS) is chosen to be the default for the system. This is the format you will normally receive when you FETCH columns of the datetime types. However, you may pass data to SQL in any of the forms, and you may FETCH data in any of the forms by using special keywords in your SQL query.

Figure 1 on page 4 demonstrates the use of the new datatypes. Note that while the data is input in the USA form, the output is in ISO form, indicating that ISO was the form chosen at SQL installation time.

```
      )LOAD 2 SQL
SAVED 1987-10-20  3.05.59 (GMT-8) 1017K(493K)
5668-899 (C) COPYRIGHT IBM CORP, 1984, 1987.
LICENSED MATERIAL - PROGRAM PROPERTY OF IBM
      )IN BIRTHDAY

      CREATE_BIRTHDAY
CREATE TABLE BIRTHDAY
(NAME CHAR(13) NOT NULL,
 BIRTHDATE DATE,
 BIRTHTIME TIME,
 ENTRYDATE TIMESTAMP)

      SQL CREATE_BIRTHDAY
 0 0 0 0 0

      INSERT_BIRTHDAY
INSERT INTO BIRTHDAY
VALUES (:1,:2,:3,CURRENT TIMESTAMP)

      DATA_BIRTHDAY
 MILLAR    06/15/1956 10:15 AM
 WHEELER   09/01/1962 06:32 AM
 WHEATLEY  09/29/1914 11:14 PM
 BROWN     12/23/1960 04:47 PM
```

Figure 1 (Part 1 of 2). Use of new datetime datatypes.

```
      SQL INSERT_BIRTHDAY DATA_BIRTHDAY
0 0 0 0 0

      2⊃SQL 'SELECT * FROM BIRTHDAY'
MILLAR         1956-06-15 10.15.00 1987-11-02-12.58.49.870681
WHEELER        1962-09-01 06.32.00 1987-11-02-12.58.50.162099
WHEATLEY       1914-09-29 23.14.00 1987-11-02-12.58.50.170724
BROWN          1960-12-23 16.47.00 1987-11-02-12.58.50.177239

      OLDESTQ
SELECT NAME,BIRTHDATE FROM BIRTHDAY ORDER BY BIRTHDATE

      SQL OLDESTQ
0 0 0 0 0    WHEATLEY       1914-09-29
             MILLAR         1956-06-15
             BROWN          1960-12-23
             WHEELER        1962-09-01

      RETIREQ
SELECT NAME FROM BIRTHDAY
WHERE YEAR(CURRENT DATE-BIRTHDATE) > 65

      SQL RETIREQ
0 0 0 0 0    WHEATLEY
```

Figure 1 (Part 2 of 2). Use of new datetime datatypes.

The APL2 SQL interface, AP 127, as shipped with APL2 Release 2, supports the use of datetime types in CREATE, and insertion of character data into tables with columns of those types. It does not, however, support FETCH of datetime data. An APAR fix is available for APL2 Release 2 which adds support for the FETCH of date/time types.

APL2 Release 3, of course, also contains that fix. In addition, three new functions (*DATE*, *TIME*, and *TIMESTAMP*) have been added to the *SQL* workspace which will convert numeric data in APL ⎕*TS* form into character strings suitable for SQL date/time columns. Figure 2 on page 6 shows examples of the use of the new functions. The left argumment of the *DATE* and *TIME* functions indicates the desired standard form.

```
      )LOAD 2 SQL
SAVED 1987-07-29 16.09.50 (GMT-8)
5668-899 (C) COPYRIGHT IBM CORP, 1984, 1987.
LICENSED MATERIAL - PROGRAM PROPERTY OF IBM

      TIME ⎕TS
13.09.23

    0 TIME ⎕TS      ⍝ 0 indicates ISO
13.09.36

    1 TIME ⎕TS      ⍝ 1 indicates USA
01:09 PM

    2 TIME ⎕TS      ⍝ 2 indicates EUR
13.09.40

    3 TIME ⎕TS      ⍝ 3 indicates JIS
13:09:44

    0 DATE ⎕TS
1987-11-02

    1 DATE ⎕TS
11/02/1987

    2 DATE ⎕TS
02.11.1987

    3 DATE ⎕TS
1987-11-02

      TIMESTAMP ⎕TS       ⍝ TIMESTAMPS are always ISO
1987-11-02-13.10.08.863

      DATE 1987 9 1       ⍝ and ISO is the default
1987-09-01
```

Figure 2. New SQL workspace functions support datetime datatypes.

In APL2, therefore, there are two ways to place the current date or time into a DB2 table. Either use the provided functions to format ⎕*TS* into a character string, or use the SQL keyword CURRENT in your SQL statement.

Single Precision Floating Point

As stated before, use of the new REAL datatype to store floating point numbers in DB2 can save space if the extra accuracy provided by double precision format is not required. Note, however, that since APL2 stores all floating point numbers in the workspace in double precision form, that space will be saved only in the database. The single precision numbers will be converted by DB2 to double precision as they are transferred to APL2. Similarly, double precision numbers will be converted to single precision by DB2 as necessary when inserting data from APL2 into SQL.

The APL2 SQL interface, AP 127, as shipped with APL2 Release 2, supports the use of the REAL datatype in CREATE, and insertion of numeric data into tables with REAL columns. It does not, however, correctly support FETCH of single precision floating point. The APAR fix previously mentioned corrects that problem.

The REAL datatype is supported only in the DB2 (TSO) environment.

VARCHAR Column Size

One of the problems noticed most often by APL2 programmers using SQL/DS is that use of the LONG VARCHAR column type can cause problems with insufficient storage. The reason for that problem is that all LONG VARCHAR columns have a maximum width of 32,767. Programs reading LONG VARCHAR data, not knowing the actual size of that data, must allocate the full 32,767 bytes for reading it. This problem is compounded in APL2 by the fact that the APL2 SQL processor normally fetches more than one row per call. The storage allocation in that case must be 32,767 multiplied by the number of rows to be fetched.

In DB2, the VARCHAR datatype allows a programmer to explicitly declare a width up to the page size chosen for the DB2 installation (32K or 4K), thus making LONG VARCHAR unnecessary. In SQL/DS Version 1, however, the maximum size for VARCHAR was 254. Any columns that were to contain data longer than 254 were forced to be LONG VARCHAR. Thus, even if the average size of data was much smaller than 32K, storage was allocated as if the data was the full 32,767 characters wide.

In SQL/DS Version 2, the restriction on size of VARCHAR columns has been lifted. There are still processing restrictions on VARCHAR columns wider than 254, as there are on LONG VARCHAR columns, but the storage allocation problem can be greatly alleviated by using VARCHAR with an explicit length declaration where LONG VARCHAR was previously used.

Figure 3 on page 8 shows a specific example of the change in storage requirements. In addition, since fewer calls to the APL2 SQL processor are required to FETCH all of the data, performance is likely to show improvement as well.

```
GIVEN:     DATA ranges from 1 to 1000 bytes
           Available space is 100,000 bytes
           Table has 1000 rows

BEFORE:    Use LONG VARCHAR
           Allocate 32K bytes per row
           FETCH 3 rows per call
           Need 334 calls to get data

AFTER:     Use VARCHAR(1000)
           Allocate 1000 bytes per row
           FETCH 100 rows per call
           Need 10 calls to get data
```

Figure 3. Effect of VARCHAR extensions on storage allocation

Database Switching in SQL/DS

SQL/DS has the capability to run multiple databases in multiple user machines. You can switch databases without leaving the APL2 environment.

With SQL/DS Version 1, you switch databases with the SQLINIT command. You can issue SQLINIT from APL2 using AP 100 or)HOST. However, before doing so, you must retract the variable shared with AP 127 to clear the interface control blocks.

In SQL/DS Version 2, if you also have APL2 Version 1 Release 3 and the VM/SP5 TSAF facility, you can switch databases by using the new extended version of the CONNECT command. The syntax for the extended CONNECT, as used from APL2, is

```
'CONNECT' 'userid' 'password' ['database']
```

which corresponds to the SQL command

```
CONNECT userid IDENTIFIED BY password [TO database]
```

If you wish to use the default userid and password, you may issue

```
'CONNECT' '' '' 'database'
```

which corresponds to the SQL command

```
CONNECT TO database
```

In addition, issuing the CONNECT with no parameters at all will query the current connection. The information will be placed in the SQLCA control block, which is available to APL2 programs by issuing the AP 127 *'MSG'* command with an argument of `(0 0 0 2 0)`.

In order to use this extended version of CONNECT, some local modifications are necessary to the APL2 SQL processor. See the installation section of this paper for details.

Isolation Level Switching

Isolation Level is a property which determines how locks are released in SQL. The **repeatable read** (RR) isolation level, which is the default isolation level in both SQL systems and in APL2, insures integrity of the data by not releasing locks until the end of the unit of work (COMMIT or ROLLBACK). Thus, ISOL(RR) guarantees that if you repeat the read, the data will be the same as it was the first time you read it. No one can modify data you have accessed.

While ISOL(RR) guarantees data integrity, it can also cause performance problems when many users are accessing SQL data concurrently. They may spend a majority of their time waiting for locks to be released, rather than doing actual data processing. The **cursor stability** (CS) isolation level can alleviate that problem.

With ISOL(CS), locks are released as the user moves the cursor off of the data, if the data has not been modified. This frees the data for access by other users.

In many applications, a large part of the use of SQL is read-only. In those cases, where data integrity is not a concern, use of ISOL(CS) may be sufficient. However, when doing updates, ISOL(RR) may be preferred to protect data integrity.

In previous releases of APL2, the isolation level was a property of the APL2 installation (CMS) or session (TSO). Thus, an application could not change isolation levels while running.

In APL2 Release 3, a new AP 127 command, '*ISOL*', is provided that allows an application to change isolation levels during program execution. This allows the programmers to provide any desired mix of isolation levels in their applications.

In TSO, the new isolation level will take effect at the beginning of the next unit of work (after COMMIT or ROLLBACK). In CMS, the new isolation level is in effect for any cursors prepared after the change is made.

See Figure 4 on page 10 for an example of the use of the ISOL command.

```
      )LOAD 2 SQL
SAVED 1987-07-29 16.09.50 (GMT-8) 505K(1017K)
5668-899 (C) COPYRIGHT IBM CORP, 1984, 1987.
LICENSED MATERIAL - PROGRAM PROPERTY OF IBM


      ISOL ''
 0 0 0 0 0  RR

      ISOL 'CS'
 0 0 0 0 0

      ISOL ''
 0 0 0 0 0  CS
```

Figure 4. Forms of the ISOL command

Double-byte Character Sets

In APL2 Version 1 Release 3, the following enhancements have been made to AP 127 in support of DBCS (double-byte character set) data:

- Users may use extended characters in SQL statements. The statements will be converted to standard SO/SI format and passed to SQL.
- Users may pass extended character data in the value-list on an OPEN or CALL command to AP 127. If the data contains DBCS characters, it will be converted to SQL GRAPHIC data before passing it to SQL. If it contains no DBCS characters, it will be converted to SQL character data.
- GRAPHIC data in SQL tables will be retrieved into APL2 as extended character data.
- Retrieval of AP 127 messages translated to DBCS languages is also supported. The translations of the messages, however, are not included with APL2.

Extended DESCRIBE

In APL2 Release 2, when an AP 127 DESCRIBE command is issued, the column names and SQL datatypes are returned.

In APL2 Release 3, the DESCRIBE command and its result have been enhanced to provide more information.

The DESCRIBE command now has an optional second parameter:

```
'DESCRIBE' name ['NAMES'|'LABELS'|'ANY'|'BOTH']
```

This syntax is compatible with the SQL syntax for DESCRIBE. If no third parameter is passed, *'NAMES'* will be used as the default.

The result of the DESCRIBE command is a 3 or 4 row matrix in the following format:

```
Row 1: If 'NAMES' or 'BOTH', the column names
       If 'LABELS', the column labels
       If 'ANY', a mixture of labels and names,
       with names where no labels exist

Row 2: Datatypes

Row 3: Null status
       If nulls allowed, the character string 'NULL'
       If nulls not allowed, the string 'NOT NULL'

Row 4: If 'BOTH', the column labels.
       If any other, this row is not returned.
```

See Figure 5 for an example of the use of DESCRIBE. In that example, the *SQL* workspace function *DESC* is used to issue the DESCRIBE commands.

```
      )LOAD 2 SQL
SAVED 1987-07-29 16.09.50 (GMT-8) 505K(1017K)
5668-899 (C) COPYRIGHT IBM CORP, 1984, 1987.
LICENSED MATERIAL - PROGRAM PROPERTY OF IBM

      PREP 'ITL' 'SELECT BIN, YEAR, TYPE, COST FROM CELLAR'
 0 0 0 0 0

      DESC 'ITL' 'NAMES'
 0 0 0 0 0    BIN       YEAR  TYPE      COST
              C 3       S     V 12      D 7 2
              NOT NULL  NULL  NOT NULL  NULL

      DESC 'ITL' 'LABELS'
 0 0 0 0 0    BIN COLUMN LABEL
              C 3                 S     V 12      D 7 2
              NOT NULL            NULL  NOT NULL  NULL

      DESC 'ITL' 'ANY'
 0 0 0 0 0    BIN COLUMN LABEL  YEAR  TYPE      COST
              C 3                 S     V 12      D 7 2
              NOT NULL            NULL  NOT NULL  NULL

      DESC 'ITL' 'BOTH'
 0 0 0 0 0    BIN                 YEAR  TYPE      COST
              C 3                 S     V 12      D 7 2
              NOT NULL            NULL  NOT NULL  NULL
              BIN COLUMN LABEL
```

Figure 5. Forms of the DESCRIBE command

Prototypes for Empty Results

In previous releases of APL2, when a FETCH returned no rows, the data item of the result of that FETCH was returned as a simple character null.

In APL2 Release 3, when a FETCH returns no rows, the data item is returned as a null prototype of the result table. As a result, if a query is repeated with different parameters, the shape of the result will be the same on each iteration, regardless of whether the result of that iteration contains any rows.

If the AP 127 **matrix** form is being used, the null result will be a matrix with zero rows and a column for each column in the table.

If the **vector** form is in effect, the result will be a vector of null items, one per column. Each of the null items is a matrix whose type and shape is the same as that of the table column it represents.

Figure 6 on page 13 shows an example of the null result for each form.

```
      NULLQ
SELECT * FROM CELLAR WHERE COLOR = 'Q'

      SETOPT 'MATRIX'
 0 0 0 0 0

      NULLRES←2⊃SQL NULLQ

      DISPLAY NULLRES
.→-------------------------.
⌽ .⊖. .⊖. .⊖. .⊖. .⊖. .⊖. |
| | | | | | | | | | | | | |
| '_' '_' '_' '_' '_' '_' |
'∊-------------------------'
      ⍴NULLRES
0 6

      SETOPT 'VECTOR'
 0 0 0 0 0

      NULLRES←2⊃SQL NULLQ

      DISPLAY NULLRES
.→---------------------------.
| .→--. .→. .⊖. .⊖. .→. .→. |
| ⌽   | ⌽0| ⌽ | ⌽ | ⌽0| ⌽ | |
| '---' '~' '_' '_' '~' '_' |
'∊---------------------------'
      ⍴NULLRES
6
      ⍴¨NULLRES
 0 3  0 1  0 0  0 0  0 1  0 1
```

Figure 6. Prototypes of empty results

User Forum

In this section we will discuss some of the most frequently asked questions about using APL2 and SQL together.

Q: What is the simplest way for an application to test whether the database connection can be successfully made?

Issue a ROLLBACK command. In previous releases, if no work had been done in the database, a ROLLBACK command did not always access the database. In APL2 Release 3, the connection logic has been made consistent across the environments. Issuing a ROLLBACK will cause a database call to be made. Since no real work has been done, if the ROLLBACK fails, the application can be fairly certain that the problem is with establishing the database connection and issue appropriate error messages to the users.

Q: If the database connection cannot be made, or the database goes down while I am processing, can I retry the connection without leaving APL2?

In most cases, retry is possible. You must retract the variable that is shared with AP 127 before doing so, in order to clear out the interface control blocks.

In SQL/DS, if a CMS OPEN error occurs, usually because the SQL/DS minidisk is not linked, SQL/DS may leave a CMS FILEDEF in effect. In that case, the FILEDEF must also be cleared before retry.

Q: We have discussed switching databases in SQL/DS. Can I switch DB2 subsystems while in APL2?

DB2 does not support the CONNECT command, so there is no SQL statement available in DB2 to provide subsystem switching. It would be a logical extension to AP 127 to provide a simulation of the CONNECT command for the DB2 environment, but no such capability exists today.

Currently, the DB2 subsystem is an APL2 invocation option. Therefore, to change subsystems, one must exit from and reenter APL2. This could be done under program control by using the stack processor (AP101), but the DB2 environment would then be terminated and would need to be reestablished.

Q: Is there any way for me to estimate the cost of a query before executing it?

After a successful SQL PREPARE, a cost estimate number is placed in the SQLCA control block by SQL. The SQLCA is available to APL2 programs by issuing the AP 127 `'MSG'` command with the argument `(0 0 0 2 0)` immediately after the prepare. Figure 7 on page 15 shows an example of this number, which is available only in APL2 Version 1 Release 3.

The SQL/DS publications state that you can use this number to determine expected relative performance of the prepared SQL statement. However, the value is relative, not absolute, and should be used with caution. It is probably better to use other techniques, such as the EXPLAIN command and the APL2 timing facility, to optimize your queries.

```
      QUERY
SELECT * FROM CELLAR
WHERE COLOR = 'R'
AND YEAR < 1980

      DAT←'PREP' 'QUERY' QUERY

      DAT
 0 0 0 0 0

      DAT←'MSG' (0 0 0 2 0)

      DISPLAY 2⊃DAT
.→-----------------------.
↓ .→.                    |
| |0|                    |
| '~'                    |
| .⊖.                    |
| | |                    |
| '-'                    |
| .→-------.             |
| |        |             |
| '--------'             |
| .→-------------------. |
| |0 0 0 1101480820 0 0| |
| '~-------------------' |
| .→-------.             |
| |        |             |
| '--------'             |
| .→-------.             |
| |        |             |
| '--------'             |
'∊-----------------------'
```

Figure 7. Query Cost Estimate

Q: What can I do if I issue a query that is taking too long, and I want to interrupt SQL processing?

In the CMS environment, press the PA2 key. If this action takes you to CMS (VM READ), it means AP 127 was waiting for SQL/DS. Type **SQLHX** to halt SQL/DS processing, or press the enter key to resume processing. If pressing PA2 causes you to suspend in APL2, AP 127 was not waiting for SQL/DS. The interrupt is handled in the normal APL2 manner.

In the TSO environment, press the PA1 key twice. You should suspend in APL2. Since TSO is an asynchronous environment, to stop DB2 processing you must also retract the variable shared with AP 127. The DB2 request will then be cancelled.

Q: Can I limit the amount of time spent in a call to SQL?

In the TSO environment, it is possible to control the time spent in DB2 by using the APL2 shared variable control facilities. By setting appropriate **access control** and using the shared variable system functions you can control how long you wait for a shared variable specification by AP 127. You then can cancel the request by shared variable retraction if it takes longer than the limit you have set. See the APL2 System Services Reference for a complete discussion of shared variable concepts.

In the CMS environment, one can use that same code without error, but since the environment is synchronous, it will not have the same effect on processing time.

*Q: I know I can call FORTRAN and Assembler programs from APL2.
Can those programs contain SQL calls?*

Yes, FORTRAN and Assembler programs called from APL2 can have SQL calls in them. This is a way to use STATIC, or compiled, SQL from APL2. However, certain cautions must be given.

First, in the DB2 environment, programs to be called from APL2 must use the **Call Attach Facility** to access DB2 rather than the DSN processor. The DSN processor requires that DB2 be in control of the environment. Since APL2 is in control of the environment when it is running, use of DSN would cause conflicts between the two. The Call Attach Facility is documented in the DB2 Advanced Application Programmer's Guide.

Second, before calling external routines that use SQL, you should terminate all SQL units of work being processed with AP 127 via ROLLBACK or COMMIT. Mixing calls to SQL through AP 127 and through an external routine will have different results in CMS and TSO, and can cause problems in both.

*Q: What is the difference in SQL between a NULL and a character
string of length 0?*

To the APL programmer, the idea that SQL distinguishes between the NULL and the 0-length string may seem rather strange. In APL the definition of a NULL character vector is that it is a string of length 0.

In SQL, however, NULL is the absence of data, and a 0-length string is data that exists, but has no dimension. The two are treated differently in queries and in sorting. For example,

```
SELECT * FROM TABLE WHERE COLUMN IS NULL
```

selects NULL items, and

```
      SELECT * FROM TABLE WHERE COLUMN = ''
```

selects 0-length strings.

This difference has created a problem in when a programmer wishes to prepare a general-purpose query,

```
        SELECT * FROM TABLE WHERE COLUMN = :1
```

where the placeholder will sometimes be replaced by a value and sometimes by a null. Since SQL has required that NULL be used only in the literal manner, with IS, the general purpose query will not work for the null case.

In SQL/DS Version 2, this restriction has been removed in compliance with the ANSI SQL standard. You can use the host varable to pass a null in that case. In DB2 Release 3, however, the restricion still applies.

The described problem holds true for all programming languages. In APL2, however, the problem is compounded by the fact that AP 127, and therefore APL2, treats the NULL and the 0-length string in the same manner - as an APL null. When fetching data into APL2, both are converted to nulls. One cannot tell which were NULL and which were 0-length. When passing data to SQL from APL2 with placeholders and value-lists, NULL is always used. The only way to use 0-length strings is to code the literal `''` into the SQL query, as shown in the example.

Several proposals have been made for ways to allow AP 127 to provide a way to distinguish these two different entities. As of this writing, a solution has not been arrived at which is both easy to use and compatible with the releases now available.

Q: What can I do to speed up my APL2 SQL programs?

There are numerous ways to tune APL2 and SQL applications. Some of the most useful will be listed here.

- Use the APL2 timing facility, available in APL2 Version 1 Release 3, to tune the APL portion of your application as much as possible.
- (SQL/DS Only) Try the BLOCK parameter on SQLPREP, if not already in use.
- Experiment with ISOL(CS) vs. ISOL(RR) in various scenarios. Caution: in a single-user environment, ISOL(RR) is faster, because less CPU time is used for it. To see the benefit of ISOL(CS), which is reduced **elapsed** time due to shorter waits, you must have multiple concurrent users.
- Vary the number of rows selected on each FETCH to minimize the number of AP 127 calls necessary to get the data.
- Make sure you have appropriate indices on your SQL tables.
- Optimize your queries. You can use the SQL EXPLAIN command to analyze query performance.
- Try the AP 127 **vector** form rather than the **matrix** form. In addition to saving space, it may also save you time.

- Do not repeat PREPARE requests unnecessarily. If you need to issue the same query with different parameters, PREPARE once with a placeholder and pass the fill-in value as you execute, rather than coding a literal string for the value in the SQL statement.
- Learn the AP 127 commands and use them directly rather than using the *SQL* function. You can often write more efficient code than the general purpose function allows for.
- Read more about SQL in the documentation for the database you are using. SQL is easy to use, and thus easy to use poorly. The more you know, the better your programs will be.

Appendix A. Installation Instructions

Instructions are provided in the APL2 Installation Manuals for preparing SQL for access by APL2. However, since this preparation requires a deviation from the default install process, and the nicest way to read a manual is not to read it at all, the necessary steps will be repeated here, with extra detail.

Note that these instructions are for APL2 Version 1 Release 3 only.

SQL/DS (CMS) Environment

To install the APL2 SQL interface:

1. LINK and ACCESS the minidisk in the SQL/DS machine. This is normally the 195 disk.

   ```
   LINK SQLDBA 195 195 RR
   ACC 195 Q
   ```

2. Initialize your user machine to the database by issuing the SQLINIT EXEC.

   ```
   SQLINIT DBNAME(SQLDBA)
   ```

   The SQLINIT process places onto your A-disk some files which are the SQL/DS "bootstrap" modules. These modules determine which SQL/DS database will be accessed. Once the files are created, there is no need to repeat this step unless they are erased or you wish to change to a different database.

3. (Optional) Modify the AP2V127I ASMSQL source module to add support for the SQL/DS Version 2 CONNECT extensions. Instructions for the modification are contained in the source module commentary.

   This step is necessary ONLY if you have SQL/DS Version 2 and VM/SP 5, and you wish to use multiple databases.

4. Preprocess the APL2 interface module. This step "registers" the APL2 program in SQL/DS.

   ```
   SQLPREP ASM PP(NOPUNCH,NOPRINT,PREP=AP2VSQL3),
           ISOL(USER),BLOCK,USER=SQLDBA/sqldbapw)
           IN(AP2V127I ASMSQL fm)
   ```

   or, if you have modified AP2V127I ASMSQL,

   ```
   SQLPREP ASM PP(NOPRINT,PREP=AP2VSQL3),
           ISOL(USER),BLOCK,USER=SQLDBA/sqldbapw)
           IN(AP2V127I ASMSQL fm)
           PU(AP2V127I ASSEMBLE fm)
   ```

   The SQLPREP command must be issued as shown, with the appropriate password and filemodes filled in. The one exception is the BLOCK parameter, which is optional. Since the use of BLOCK provides significant performance improvements, it is highly recommended. Some SQL queries, however, do not work when BLOCK is in effect. For more information on BLOCK, see the SQL/DS Application Programmer's Guide.

5. Grant authority for users to use the APL2 interface.

```
CONNECT SQLDBA IDENTIFIED BY sqldbapw
GRANT RUN ON AP2VSQL3 TO PUBLIC
```

   The CONNECT and GRANT commands can be issued using the ISQL processor, or with the SQLDBSU EXEC.

6. (Optional) If you have modified AP2V127I ASMSQL, and created AP2V127I ASSEMBLE, assemble the module. Assembler H is required. The AP2MAC MACLIB must be available, and a GLOBAL MACLIB AP2MAC statement issued before running the assembly.

   Place the resulting AP2V127I TEXT file on a disk that precedes the APL2 installation disk in the CMS search order, and generate the APL2 load module according to the instructions provided in the CMS installation guide. The installation process will replace the default AP2V127I TEXT with your modified one.

If the installation has multiple SQL/DS databases, and APL2 will be used from each of them, steps 1 through 5 must be repeated for each database.

To use the APL2 SQL interface:

1. See your database administrator to be granted appropriate authority to access the database and obtain space for creating tables. There are different levels of authority. For more information, see the SQL/DS Planning and Administration Guide.
2. LINK and ACCESS the SQL/DS minidisk, as in step one above.

   If you will be using SQL and APL2 together frequently, you may want to add the LINK and ACCESS to your AP2EXIT EXEC so it is done automatically each time you invoke APL2.
3. Issue SQLINIT as in step 2 above, if the bootstrap modules do not already exist on your A-disk.
4. Invoke APL2

DB2 (TSO) Environment

To install the APL2 SQL interface:

After the SMP APPLY step for the APL2 product, run the AP2JBIND job. This will bind and authorize users to run the APL2 application plans.

If the installation has multiple DB2 subsystems, and APL2 will be used from each of them, the above steps must be repeated for each subsystem.

To use the APL2 SQL interface:

1. See your database administrator to be granted appropriate authority to access the database and obtain space for creating tables. There are different levels of authority. For more information, see the DB2 Reference Manual.

2. ALLOCATE the DB2 load libraries in your TSO LOGON PROC or your APL2 CLIST.
3. Invoke APL2

Appendix B. References

DB2 Publications

1. IBM Database 2 SQL Learner's Guide (SC26-4082)
2. IBM Database 2 Application Programming Guide (SC26-4293)
3. IBM Database 2 Advanced Application Programming Guide (SC26-4292)
4. IBM Database 2 SQL Reference (SC26-4346)
5. IBM Database 2 Reference Summary (SX26-3740)
6. IBM Database 2 Messages and Codes (SC26-4113)

SQL/DS Publications

1. SQL/DS Database Planning and Administration (SH09-8017)
2. SQL/DS System Planning and Administration (SH09-8018)
3. SQL/DS Application Programming Guide (SH09-8019)
4. SQL/DS Messages and Codes (SH09-8052)

APL2 Publications

1. APL2 Programming: Using Structured Query Language (SH20-9217)
2. APL2 Programming: Guide (SH20-9216)
3. APL2 Programming: System Services Reference (SH20-9218)
4. APL2 Reference Summary (SX26-3737)

Other

1. Development Guide For Relational Applications (SC26-4130)
2. Techniques in SQL Application Design (TR 03.290)
3. Interactive SQL and APL2 (TR 03.289)