CMS PIPELINES: THE SQL STAGE

August 15, 1995

Nick Laflamme

Office of University Computing
University of Notre Dame

The Genix Group
Dearborn, MI
IBMMAIL: USMCNJRN

Session 9112

Abstract

Both SQL/DS and CMS Pipelines are wonderful tools for working with records of data, and thanks to CMS Pipelines' *sql* stage, you can mix SQL and Pipes tools to load and unload data from your SQL tables. How do you convert your data from Pipes formats to SQL formats and back again? Do you use SQL to filter out unwanted data, or do you let Pipes do it for you? Look at the easy answers as well as the pros and cons of the more complex questions. Gain a better grasp of the complexities so you can come away with your own counter-arguments. Gain, too, a list of warnings about possible errors. Most of all, gather some ideas about how to mix two of your favorite data manipulation tools!

Preface

I'd like to thank my previous employer, the University of Notre Dame, for giving me the chance and the tools to do the work described in this talk.

I'd like to thank my current employer, the Genix Group, for allowing me to give this talk.

Table of Contents

Background

A long time ago, on a 4341 in a research lab sponsored by IBM Endicott, I learned to be a VM Systems Programmer. They made sure we had lots of VM software, and I tried to make it useful for some student researchers. One day I wandered into SQL/DS and taught myself some, enough to be dangerous, maybe even useful.

Then I graduated and went to work in a place that had CMS Pipelines, before real customers had CMS Pipelines. My SQL/DS skills, to use the term loosely, languished from disuse, but I acquired some rudimentary CMS Pipelines skills.

Eventually I moved on to build a VM system for a university, having fairly easy access to lots of IBM software because this was an academic system, not an administrative system. I actually had a faculty request for an SQL server almost immediately, so I dusted off my SQL/DS skills and eventually used SQL/DS for some projects of my own.

Eventually IBM released CMS Pipelines (no, in a fit of integrity, I hadn't taken a tools tape with me when I left Endicott), and eventually, I got real tired of jumping through hoops with RxSQL.

You'll notice:

> I'm a VM Systems Programmer who finds find SQL/DS useful for some things, not a trained SQL/DS administrator or SQL programmer.

Before making any financial decisions based on this talk, consult your accountant or financial advisor to see if any of this applies to you. You'd better double check what I say with an SQL/DS programmer, too.

Note: Although this talk refers to SQL/DS servers, it probably is equally applicable to DB/2 servers, either from CMS Pipelines using APPC kinds of conversations or from TSO Pipelines if such things exist.[1]

[1] Yes, such things exist, I'm just being coy.

Sources and Credit

A lot of this talk is based on what I've learned from a few specific places:

- The SQL chapters in the CMS Pipeline users' guides that I've had accessed, while not long or elaborate, tend to be filled with gems of knowledge.
- I've received more personal assistance from John Hartmann and Melinda Varian than anyone has a right to, and a lot of my questions of them have come while doing SQL stages.
- The CMSPIP-L e-mail list is a good general resource about CMS Pipelines, and the SQLINFO list is where I tend to get the SQL issues resolved. The trick is knowing when something is an SQL issue or a CMS Pipelines issue. Both are Revised LISTSERV lists.

Making It Work

Rules of the *sql* stage:

1. Whatever the question is, it probably involves the *spec* stage.
2. SQL is still SQL. The commands are the same, but the data may come and go in a different flavor than expected.

If you know how to phrase a **SELECT** statement in SQL, it's not any different using CMS Pipelines. If you know how to **INSERT** data into SQL, it's not really any different using CMS Pipelines. Most of the tricks come in understanding the format of the data as it comes and goes.

The *sql* stage can also be used to issue commands. However, since CMS Pipelines is so oriented to handling data, so my uses of the *sql* stage are oriented toward handlng data, not commands. (On the other hand, writting a command line PIPE is often faster than going into ISQL or QMF to execute some SQL command!)

Reading Data

What good is it to have data in an SQL/DS table unless you can read it? Just as money in the bank is only useful if you can withdraw it, so, too, is data in an SQL/DS table worth anything only if you can read it.

To some of us, of course, data is only considered readable if we can read it using CMS Pipelines.

What Have We Here?

If you know how SQL/DS regards a table or view, it's not hard to guess how CMS Pipelines might regard the results of a query against the data.

Still, why guess? Use the **DESCRIBE** operand for **SELECT** to find out just what CMS Pipelines finds about a query.

```
pipe sql describe select docno,pub_date,rev_date,title
from docreqs.xdocno_title | cons
452      CHAR            8     8 DOCNO
385      DATE           10    10 PUB_DATE
385      DATE           10    10 REV_DATE
449      VARCHAR       128   130 TITLE
```

Note that “TITLE” has a two byte prefix indicating just how long a record's title field really is. What isn't indicated directly indicated is whether a field might be null, although that *might* be indicated in the three digit prefix for each field.

Instead, unless you specify **NOINDICATOR** on your **SELECT**, CMS Pipelines prefixes each field with a halfword saying if the field in that record is null or not.

While **DESCRIBE** probably can be used to dynamically build a PIPE to process some SQL queries, I content myself to use it when writing PIPEs by hand.

I Want It!

OK, so you know what kinds of data to expect.

```
PIPE SQL SELECT DOCNO,PUB_DATE,REV_DATE,TITLE
         FROM DOCREQS.XDOCNO_TITLE
| spec 3.8 1 25.10 10 37-* 21
| cons
```

You take the data, you strip off those null indicators because you forgot **NOIND**, you ignore a field you decide you don't need, and there it is!

There's A Piece Missing!

What if some of those revision dates are missing? do you want to use the publication dates in their stead?

```
PIPE (end \) SQL SELECT DOCNO,PUB_DATE,REV_DATE,TITLE
         FROM DOCREQS.XDOCNO_TITLE
| sql execute ',
| a: locate 23.2 /'||'0000'x||'/', /* find revision dates */
| spec 3.8 1 25.10 10 37-* c2v 21
| b: faninany
| cons
\ a:
| spec 3.8 1 13.10 10 37-* c2v 21
| b:
```

If the indicator before the field at column 25 says you've got data, use it! Otherwise, simply aim *spec* at the publication date!

You can do this with joined queries in SQL, but to me, this is easier.

Funny Datatypes

It's too much to hope that all your data is inherently human readable, even if CMS Pipelines tends to treat data as human readable. On the other hand, if SQL/DS knows data is numeric, you can bet it'll be stored for efficiency, not for people eyeballing dumps.

Remember that rule of the *sql* stage?

- Whatever the question is, it probably involves the *spec* stage.

spec knows how to translate real numbers into human readable representations. It knows how to translate integers into human readable form. It knows how to use the length prefix for a VARCHAR so your data isn't unnecessarily long. It probably can perform minor surgical procedures, but we'll leave that for another time.

- C2D
- C2P
- C2V

correspond to:

- Display decimal data
- Display packed decimal data
- Convert variable character strings

Inserting Data

SQL/DS databases, like bank accounts, are initially empty. Therefore, one needs to be able to load data into a database or update data in a database so that one can later retrieve it.

The good news is, loading data into an SQL/DS table with CMS Pipelines is just like reading data from an SQL/DS table with CMS Pipelines.

The bad news is, loading data into an SQL/DS table with CMS Pipelines is just like reading data from an SQL/DS table with CMS Pipelines.

You know how to do it by analogy, but no one says it's easy or simple.

A Gruesome Example

Let's just jump in, shall we?

```
PIPE (end \) firststage,
| b: find VM/ESA IPOE|
  specs 106.8 1 /19/ 10 60.2 12 /-/ 14 54.5 15
| change %/%-%
| a: juxtapose
| timemins 45 54
| timemins 35 43
| timemins 56 65
| specs 1.8 1 10.10 9 21.8 19 /* account, date, userid */
        30.4 d2c 27.2 right                /* sessions */
        pad 00           /* pad with nulls, not spaces */
        35.9 p2c(2) 29.5 right             /* connmins */
        45.10 p2c(2) 34.5 right               /* Tmins */
        56.10 p2c(2) 39.5 right                /* Vmins*/
        67.9 d2c 44                            /* SIOs */
| SQL NOINDICATORS INSERT INTO NLAFLAMM.VMUSEHISTORY
  (ACCOUNT, EOFMONTH, USERID, SESSIONS, CONNMINS,
   TOTMINS, VIRTMINS, SIO),
| >> LOADACCT HISTORY A
\ b:
| nfind PROJ TOT|
  nfind Bad cards skipped:|
  nfind  TOT|
  nlocate 10.4 /SESS/
| specs 1.23 2 31.21 26 73.9 48
| a:
```

Is that a gruesome SPECS stage or what?

- "P2C(2)" converts the data into packed format with two decimal places.
- Integers are handled by "D2C".
- "PAD 00" ensures the numbers converted to packed have leading "00", not leading "40" characters.

Note that I specified "NOINDICATORS" on the INSERT command. If I wanted to insert null fields, I'd have to use indicator prefixes.

Nuances of SELECTs

When one combines the power of two complex tools, one often gains a multitude of ways to skin a cat. In some cases, certain techniques have clear advantages over others for some purposes. In other cases, while different techniques have different strengths and weakness, it's not clear if one technique is simply always better than another.

When composing SQL queries using CMS Pipelines, often the point to consider is, should a function be done in the SQL/DS server, or should it be done in the user virtual machine? Using more complex SQL statements forces more processing to occur in the SQL server. Using more Pipes stages, of course, would do more processing in the user's virtual machine. The costs or values of either approach are probably site dependent.

Generally speaking, SQL/DS is better at working with complete fields; CMS Pipelines is better at working with data in arbitrary locations within records, such as subsets of fields. Recent releases of CMS Pipelines include more support for fields within records, particularly with (all together now) *spec*.

In many cases, answers are expected to vary depending on whether appropriate fields in SQL/DS are indexed or not.

Concept	CMS Pipeline	SQL Select	Index sensitive?
Sorting	sort	ORDER BY	yes
Uniqueness	unique	DISTINCT	yes
Merging Results	fanin{any}	JOIN	no
Subselection	lookup collate	WHERE . . . SELECT	yes
Selection	(n)locate, (n)find, frlabel, tolabel, between, pick	WHERE	yes
Counting Hits	sort, count, unique, last	COUNT()	no
Stats and Math	take {last} n, spec	min(), max(), avg(),	

Table 1. Data Manipulation Techniques: CMS Pipeline stages and similar enhancements to SELECT statements, with an indication of whether indices on columns are likely to make differences.

PICK is a relatively new CMS Pipelines stage. *SPECS* is an older stage, of course, but its math functions are very new.

Strengths

Are there reasons to do the processing of your query using either the SELECT statement itself or additional Pipeline stages? Of course there are!

SQL

SQL/DS provides a richer array of mathmatical and statistical functions than CMS Pipelines. Built in functions like max() and min() are simpler to code and understand than the equivilent functions in CMS Pipelines.

SELECT statements also offer a wider range of comparisons for selecting records. For example, one can easily select records where a salary exceeds some figure even if none of the records have that particular value in them.

CMS Pipelines

If one is comfortable with multi-stream Pipelines, complex selections where a **SELECT** might have to use **JOIN**s or **UNION**s might be simpler using addtional Pipes stages such as *fanin* or *faninany*.

CMS Pipelines may be more tolerant of mixing data types that aren't exactly the same. After all, to Pipelines, it's all just human readable characters unless twisted and warped by SPEC.

As noted above, Pipelines will let you work within fields much more readily than SQL/DS will. You can specify column offsets within a record for stages like *sort* and *locate*, which can be an advantage over pure **SELECT** commands.

CMS Pipelines also will let you use multi-stream pipes to gather easily the debris from your selection processes. For example, if you use *lookup* to match fields between two records, you can also find out which records from each query weren't matched up with records from the other query without issuing extra queries. Doing so using native **SELECT**s would take an extra query or two depending on what information you want.

Weaknesses

There are, unfortunately, weaknesses in both tools. Arguably, these weaknesses reflect design goals to do fewer things better, not more things adequately. Still, they are areas where a programmer will have to mix in more tools or languages to accomplish tasks.

SQL

While SQL/DS **SELECT**s will support wildcard matching against character strings, they're fairly dismal for case insensitive queries.

It's also too easy sometimes for a programmer to write a query that SQL/DS for some reason makes very complex and long-running.SQL experts might be able to identify quickly in each case what the programmer should do to make the query more efficient and simple (in SQL/DS's perspective), but to a programmer who just inadvertantly launched the Query from Heck, that's small comfort.

If you want to take the *n* records with the highest values for some field, if *n* isn't 1, it's hard to do with an SQL/DS query.

CMS Pipelines

Even with the recent 407 emulation enhancements to the *spec* stage, using math and statistical functions in CMS Pipelines is going to generally be more complex and daunting than the same functions would be in an SQL/DS query. Similarly, selection functions are unweildy, especially when a user needs to select a range of values. Take, for example, the following word problem with the following data:

```
SELECT WHERE LASTNAME BETWEEN 'E' AND 'I'

BUCKINGHAM, LINDSEY Fleetwood Mac
CLAYTON, ADAM       U2
FLEETWOOD, MICK     Fleetwood Mac
HEWSON, PAUL        U2
McVIE, CHRISTINE    Fleetwood Mac
MULLINS, LARRY JR.  U2
NICKS, STEVIE       Fleetwood Mac
```

- Case insensitive queries are improving in recent releases of CMS Pipelines, but still relative weakness for CMS Pipelines

Summary

There truly is more than one way to skin a cat, especially when you combine CMS Pipelines with the SQL query language. With both tools you need to know the structure of your data, and from your knowledge of your data you'll get some indicators of how to solve your problems.

After that, though, a lot of discretion is left to the programmer, and the programs written will probably reflect whether the programmer is more comfortable with CMS Pipelines or SQL queries.