Craig S. Mullins

Return to Home Page

Vol. 10, No. 2 (August 2003)

From IDUG Solutions Journal...

The Buffer Pool

Choose the Proper Data Type

By Craig S. Mullins

The basic theme of this column has been to review DB2 issues that are confusing or misunderstood. This time, I'll be tackling a topic that might surprise you – using the proper data types.

Indeed, data type and length are the most fundamental integrity constraints applied to data in a database. Simply by specifying the data type for each column when a table is created, DB2 automatically ensures that only the correct type of data is stored in that column. Processes that attempt to insert or update the data to a non-conforming value will

be rejected. Furthermore, a maximum length is assigned to the column to prohibit larger values from being stored in the table.

The DBA must choose the data type and length of each column wisely. It is almost always best to choose the data type that most closely matches the domain of correct values for the column. In general, adhere to the following rules:

- If the data is numeric, favor SMALLINT, INTEGER, or DECIMAL data types. FLOAT is also an option for very large numbers only.
- If the data is character, use CHAR or VARCHAR data types.
- If the data is date and time, use DATE, TIME, and TIMESTAMP data types.
- If the data is multimedia, use GRAPHIC, VARGRAPHIC, BLOB, CLOB, or DBCLOB data types.

Now, I don't know exactly why using improper data types is such a widespread practice, but I can guarantee you that it is. I am constantly being bombarded with questions that bounce around this topic. A lot of folks have trouble

when trying access data that is not stored using a standard format or data type. Maybe they just cannot figure out how to get that function to work right. Or perhaps they're just having a performance problem. Sometimes the answer to all of these questions can be – use the proper data type!

DATE and TIME

Why anyone would ever store a date or time in a DB2 table any other format than DATE, TIME, or TIMESTAMP is beyond me. But, oh, they do it all the time. And it causes all sorts of headaches. The benefits of using the proper DB2 data type are many, including:

- Ensuring data integrity because DB2 will ensure that only valid date and time values are stored
- The ability to use date and time arithmetic
- A vast array of built-in functions to operate on and transform date and time values
- Multiple formatting choices

DB2 enables you to add and subtract DATE, TIME, and TIMESTAMP columns. In addition, you can add date and time durations to or subtract them from these columns. Understanding these capabilities and features can dramatically decrease your programming time and effort. Keep the following rules in mind:

When you issue date arithmetic statements using durations, do not try to establish a common conversion factor between durations of different types. For example, the following two date arithmetic statements are not equivalent:

```
2003/04/03 - 1 MONTH
2003/04/03 - 30 DAYS
```

April has 30 days, so the normal response would be to subtract 30 days to subtract one month. The result of the first statement is 2003/03/03, but the result of the second statement is 2003/03/04. In general, use like durations (for example, use months or use

days, but not both) when you issue date arithmetic.

If one operand is a date, the other operand must be a date or a date duration. If one operand is a time, the other operand must be a time or a time duration. You cannot mix durations and data types with date and time arithmetic.

If one operand is a timestamp, the other operand can be a time, a date, a time duration, or a date duration. The second operand cannot be a timestamp. You can mix date and time durations with timestamp data types.

Now, what exactly is in that field returned as the result of a date or time calculation? Simply stated, it is a duration. There are three types of durations: date durations, time durations, and labeled durations.

Date durations are expressed as a DECIMAL(8,0) number. To be properly interpreted, the number must have the format

yyyymmdd, where yyyy represents the number of years, mm the number of months, and dd the number of days. The result of subtracting one DATE value from another is a date duration.

Time durations are expressed as a DECIMAL(6,0) number. To be properly interpreted, the number must have the format hhmmss, where hh represents the number of hours, mm the number of minutes, and ss the number of seconds. The result of subtracting one TIME value from another is a time duration.

Labeled durations represent a specific unit of time as expressed by a number followed by one of the seven duration keywords: YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS, or MICROSECONDS. A labeled duration can only be used as an operand of an arithmetic operator, and the other operand must have a data type of DATE, TIME, or TIMESTAMP. For example:

```
CURRENT DATE + 3 YEARS + 6 MONTHS
```

This will add three and a half years to the current date. Keep in mind, though, that you cannot use a host variable for the operand of a labeled duration – it must be a numeric constant.

Additionally, you have multiple built-in functions at your disposal for manipulating date and time data, but only when the data is stored as DATE, TIME, and TIMESTAMP data types. The time-related DB2 functions include CHAR, DATE, DAY, DAYOFMONTH, DAYOFWEEK, DAYOFYEAR, DAYS, HOUR, JULIAN_DAY, MICROSECOND, MIDNIGHT_SECONDS, MINUTE, MONTH, QUARTER, SECOND, TIME, TIMESTAMP, WEEK, WEEK_ISO, and YEAR.

The way a DATE is internally stored is not really that important. However, the way dates are displayed is very important to most applications, and DB2 offers a good range of formats. Consult Table 1 for a list of the date formats that are supported by DB2. You may also have an installation-defined date format

that would be named LOCAL. For LOCAL, the date exit for ASCII data is DSNXVDTA, the date exit for EBCDIC is DSNXVDTX, and the date exit for Unicode is DSNXVDTU.

Table 1. DB2 DATE Formats

Name	Layout	Example
ISO	yyyy-mm-dd	2002-10-22
USA	mm/dd/yyyy	10/22/2002
EUR	dd.mm.yyyy	22.10.2002
JIS	yyyy-mm-dd	2002-10-22
LOCAL	Locally defined layout	N/A

The only situation when DATE, TIME, or TIMESTAMP may not be appropriate for chronological data that immediately jumps to mind is when you only need to store a subset of a date or time. For example, if all that is required is month, then a DB2 DATE or TIMESTAMP would require you to store information that is not needed and might be confusing. So you might choose to create an

INTEGER column for month with a check constraint limiting the values to 1 through 12. Other than that type of situation, DATE, TIME, or TIMESTAMP is the way to go.

What problems can happen if you choose a numeric or CHAR data type instead? Well, you will not be able to perform date arithmetic or take advantage of DB2's formatting options. Of course, you could convert the data to a DB2 DATE or TIME first, but that can cause performance problems. CPU usage is required to convert data to and from different formats. For example, what if you choose to store dates in a CHAR column in the following format: yyyymmdd (with no dashes or slashes). To use a DB2 format or function you will have to convert the data to a recognizable DB2 format. To do this, you could use the SUBSTR function to break the character column apart into the separate components and concatenate them together into the USA format as follows:

```
SUBSTR(column,5,2) || "/" || SUBSTR(column,7,2) || "/" ||
SUBSTR(column,1,4)
```

You can then use the DATE function so that the result of this can be used in date arithmetic with other dates or date durations. Of course, with all the substringing it may not perform extremely well.

So, use DATE, TIME, and TIMESTAMP for your chronological DB2 data.

NUMERIC versus CHARACTER

But chronological data is not the only area where data type troubles abound. Sometimes we can mess up quite nicely just choosing between numeric and character data types. For example, consider the following scenario. A four-byte code is required to identify an entity; all of the codes are numeric and will stay that way. But, for reporting purposes, users wish the codes to print out with leading zeroes. Should the column be defined as CHAR(4) or SMALLINT?

Without proper edit checks, inserts and updates could place invalid alphabetic

characters into the product code. This can be a very valid concern if ad hoc data modifications are permitted. This is rare in production databases, but data problems can still occur if the proper edit checks are not coded into every program that can modify the data. If proper edit checks are coded and will never be bypassed, this removes the data integrity question. Check constraints are not a very viable solution in this case, either. Consider the following constraint

```
COL  CHAR(4) CONSTRAINT NUMBER CHECK
             (COL >= '0000' AND
              COL <= '9999'),
```

This constraint would allow the following value `'0ABC'`. It is greater than `'0000'` and less than `'0000'`, so it fits within the constraint.

Choosing the wrong data type can impact performance, too. Consider the impact on filter factor calculations for numeric versus character data. What are the possible number of values that a CHAR(4) column and a SMALLINT column can assume? Even if programmatic edit checks are coded for each,

DB2 is not aware of these and assumes that all combinations of characters are permitted. DB2 uses base 37 to determine access paths for character columns, under the assumption that 26 alphabetic letters, 10 numeric digits, and a space are most commonly used. This adds up to 37 possible characters. For a four-byte character column there are 37^4 or 1,874,161 possible values.

A SMALLINT column can range from -32,768 to 32,767 producing 65,536 possible small integer values. The drawback here is that negative or 5 digit product codes could be entered. However, if we adhere to our proper edit check assumption, the data integrity problems will be avoided here, as well.

DB2 will use the HIGH2KEY and LOW2KEY values to calculate filter factors. For character columns, the range between HIGH2KEY and LOW2KEY is larger than numeric columns because there are more total values. The filter factor will be larger for the numeric data type than for the character data type, which may

influence DB2 to choose a different access path.

But what about those leading zeroes? If the data is stored in a CHAR(4) column we can just input the data with leading zeroes and then they will always be there. Well, this “problem” can be solved using other methods. When using QMF, you can ensure that leading zeroes are shown by using the "J" edit code. Other reporting tools have similar capabilities. Internally developed reporting programs can be coded to display leading zeroes easily enough by moving the host variables to appropriate display fields.

In general, once again, all signs point to assigning the column the data type that best matches the values in its domain.

Summary

There is a lot more that can be said about choosing appropriate data types, but I have run out of space for this issue. The bottom line

on data types is to use them to protect the integrity of your DB2 data and to simplify your job by taking advantage of DB2's built-in capabilities. By choosing that data type that most closely matches your data you will be doing yourself, your systems, and your users a big favor.

From *IDUG Solutions Journal*, August 2003.

Home.