Craig S. Mullins

Database Performance Management

Return to Home Page

DB2 update

November 2007

New Data Types in DB2 9 for z/OS
By Craig S. Mullins

DB2 9 for z/OS offers a plethora of new features, but sometimes what seems like one of the more mundane new features can actually be quite beneficial. Such is the case with data types. DB2 9 for z/OS offers five new data types, so let's examine each of them in a little detail.

BIGINT

First up, we have the BIGINT data type. A BIGINT is an exact numeric data type capable of representing 63-bit integers. This is the third integer data type now available to DB2 and it offers the ability to store the largest range of values:

- SMALLINT values can range from -32768 to 32767
- INTEGER values can range from -2147483648 to 2147483647
- BIGINT values can range from -9223372036854775808 to 9223372036854775807

So when you have the need to store very large integers you don't have to use DECIMAL with a zero scale any longer.

BINARY and VARBINARY

Next up, and perhaps more exciting, V9 delivers a *true* binary data type for the first time in DB2. That means we no longer have to use a CHAR with FOR BIT DATA to store binary values in DB2 tables. BINARY is a fixed-length binary string up to 255 bytes. DB2 V9 also delivers a VARBINARY data type, which is a variable-length binary string up to 32704 bytes.

BINARY and VARBINARY data types extend current support of binary strings (BLOB), and are compatible with the BLOB data type. They are not compatible with character string data types. IBM does make it somewhat easy to migrate existing columns defined as CHAR FOR BIT DATA or VARCHAR FOR BIT DATA over to BINARY or VARBINARY. If there is an index defined on the column, the index is placed in RBDP. You cannot alter BINARY or VARBINARY data types to CHAR FOR BIT DATA or VARCHAR FOR BIT DATA.

Also, there are some usage considerations to keep in mind. Two binary strings are equal only if the lengths are identical. If two strings are equal up to the length of the shorter string length, the shorter string is considered less than the longer string, even when the remaining bytes in the longer string are hex zeros.

Decimal Floating Point

The next new data type supported by DB2 9 for z/OS is DECFLOAT. V9 takes advantage of new System z9 hardware support delivering the DECFLOAT data type that lets you use decimal floating-point numbers with greater precision. A decimal floating-point value (DECFLOAT) is an IEEE 754r number with a decimal point.

The maximum precision is 34 digits and the range of a decimal floating point number is either 16 or 34 digits of precision. The range of values supported by DECFLOAT is as follows:

DECFLOAT(16)	**Low**	**High**
• Range	-9.999999999999999×10**384	9.999999999999999×10**384
• Smallest Positive Value	1.000000000000000×10**-383	
• Largest Negative Value		-1.000000000000000×10**-383
DECFLOAT(34)		
• Range	-9.999999999999999999999999999999999 ×10**6144	9.999999999999999999999999999999999 ×10**6144
• Smallest Positive Value	1.000000000000000000000000000000000 ×10**-6143	
• Largest Negative Value		-1.000000000000000000000000000000000 ×10**-6143

Anyone think they'll need to store bigger numbers? I doubt it. In addition, the DECFLOAT data type is able to represent the following named special values representing "non-number numbers":

- **Infinity** - a value that represents a number whose magnitude is infinitely large.
- **Quiet NaN** - a value that represents undefined results which does not cause an invalid number condition. NaN is not a number.
- **Signaling NaN** - a value that represents undefined results which will cause an invalid number condition if used in any numerical operation.

XML

And finally, we get pureXML support to store XML as a native data type in DB2. That means you can specify XML as a data type for columns in your DB2 tables in DB2 9 for z/OS. But I'm not really going to elaborate any further on XML here. A few more details can be found in an earlier post here (http://www.db2portal.com/2007/04/native-xml-support-in-db2-databases-db2.html)

One of the biggest technological advances in DB2 V9 is the ability to combine the management of structured and unstructured data. Basically, V9 will allow you to store data as native XML. This capability has already been introduced into V9 of DB2 for Linux, Unix, and Windows.

At this point you might well ask "Hey, what's the big deal here? Can't we already use the XML Extender and store XML data in DB2 prior to V9?" Yes, but V9 changes the game. You will be able to search and analyze structured data in a relational data repository and unstructured data in an XML repository without the need to reformat it. So your regular "relational" data gets stored as always; and XML data gets stored in its native format without the need to shove it into a CLOB or shred it into "relational" columns.

The approach is novel in that DB2 will now support native XML via dual storage engines – the traditional SQL/relational engine and a new XML engine. DB2 9 for z/OS handles XML as a new data type that is stored in a natural hierarchy - different from relational data.

For those of you not familiar with XML, you need to know that there are big differences between XML data and typical DB2 data. Foremost among these differences is that XML data is hierarchical, whereas "relational" DB2 data is basically "flat."

Additionally, XML data is self-describing. XML tags identify and name the data elements in the XML document. This capability concentrates both the data and its structure into a single document. So, in essence, the XML document becomes self-describing. This is important to keep in mind because a single XML document can have many different types of data, whereas "relational" DB2 data is defined in the system catalog by its column definition. And all data in the same column must have the same data type (e.g. you cannot store a name in an integer column).

Finally, XML data is ordered, whereas “relational” DB2 data is not. The order in which data items are specified in the XML document is relevant. There is often no other way to specify order within an XML document. For relational data, the order of the rows is not guaranteed unless you specify an ORDER BY clause on one or more columns.

OK, now, just how would you support XML data in DB2 V9 then? Think of XML as just another data type. You would use the XML data type in a CREATE TABLE statement to define a column to be of type XML. Each column of type XML can hold one XML document for every row of the table. Even though the XML documents are logically associated with a row, XML and “relational” columns are stored differently. The “relational” columns are stored in the traditional structures we all know and love. The XML data is stored in hierarchical structures.

Don’t let that scare you. IBM has seamlessly integrated XML with relational data to simplify application development while optimizing search performance with highly optimized XML indexes.

Here is a quick example that creates a simple table with an XML column. First, as with triggers, when you create tables with XML in SPUFI be sure to set the SQL terminator to a character other than a semicolon, for example, the pound sign (#). This is done so that your SQL can have embedded semicolons. Also, you’ll probably want to set CAPS OFF in SPUFI to preserve lower case. Then, create a table like this:

```
CREATE TABLE MYCUSTOMER
(CID BIGINT,
 INFO XML)
#
```

This DDL creates a table with two columns, the first column as a big integer and the second for the XML data. Next, we’ll build an index over XML data. We will assume that the XML documents to be stored in the INFO column will have a root element named customerinfo with an attribute named Cid. So, here is the DDL for the unique index on the Cid attribute:

```
CREATE UNIQUE INDEX MYCUT_CID_XMLIDX ON MYCUSTOMER(INFO)
GENERATE KEY USING XMLPATTERN
‘declare default element namespace
   "http://posample.org"; /customerinfo/@Cid’
AS SQL DECFLOAT
#
```

The XML pattern defining the index is case-sensitive. The element and attribute names in the XML pattern must match the element and attribute names in the XML documents exactly.

After inserting data to the tables you can issue queries against the table and retrieve the XML data. V9 also supports XPath to query elements within an XML document, as well as catalog extensions to support definitions of XML schemas. Furthermore, the IBM DB2 utilities have been extended such that they can be used to administer XML data, too.

To my mind, though, one of the problems with XML in DB2 9 for z/OS is the lack of support for XQuery. XQuery is an XML query language capable of traversing XML documents. Just like SQL is the query language for native DB2 data, XQuery is the query language for native XML data. DB2 9 for Linux, Unix, and Windows supports XQuery, but DB2 9 for z/OS does not. So, how do you retrieve XML data using DB2 9 for z/OS? You can use SQL to retrieve entire XML documents from XML columns just like you would retrieve any other column. But if you need to retrieve portions of that XML document you will need to specify XPath expressions, through SQL with XML extensions.

Of course, there is a lot more to learn about XML and DB2's built-in pureXML support, but this should suffice for a quick introduction to the XML data type.

Summary

So, as you can see, just saying that DB2 provides 5 new data types understates the new functionality a bit, doesn't it? These new data types offer a significant amount of new functionality and will be quite helpful as you design and implement databases using DB2 9 for z/OS.

From DB2 Update, November 2007.

Home.