

Free Your Apps from Key Generation with a DB2 Sequence

By Kent Milligan

This article was originally published in the September, 2004 issue of System iNEWS magazine (systeminetwork.com)

Most programmers these days have no shortage of work to do. No matter how fast you can move your fingers on a keyboard, it's impossible to keep up with all the application code that needs to be written, maintained, and tested. For that reason, application developers should be looking for ways to have DB2 UDB for iSeries do some of this work for them. The Sequence object, new in V5R3, provides a great opportunity to have DB2 take care of simple application code such as key value generation.

A DB2 sequence object can not only reduce the amount of code that has to be written, but they can also make your solutions more efficient. This is due to the fact that the DB2 engine can generate and synchronize the incrementing of key values at a lower level in the operating system than your application.

Sequence vs. Identity

Sequences actually follow in the footsteps of a V5R2 enhancement, Identity columns (see *"Taking on a DB2 UDB Identity" November 2002, article 15329 at iSeriesNetwork.com*), which also allow you to have DB2 generate key values for an application. So the next logical question is "Why does DB2 need two methods for key value generation?" Let's spend a little time comparing and contrasting the two approaches.

Identity columns do a great job of generating key values for each new row that is inserted into a single table. However, a table can contain only one Identity column. So if an application has a table where DB2 needs to generate both the order number and payment number for a row being inserted into a table, it cannot be done with Identity columns since the table can have at most one Identity column. Identity columns also require DB2 to generate a new value for each row that is inserted into a table. This restriction does not work well when the application wants the same order number shared across multiple rows being inserted into an order detail table.

In contrast, a DB2 Sequence object can really be viewed as providing a superset of the capabilities offered by an Identity column. A sequence can be used to supply a new key value for every row inserted into a table, but can also go well beyond the Identity column limitations that were just discussed. It should be noted that an Identity column will always be the best performing option when an application just needs DB2 to generate a new key value for every row in a single table.

A Sequence can offer more capabilities because it centralizes the key value into a standalone DB2 object. The key value generation is no longer tied to a single table. If multiple tables all need to share the same generated key value then all the tables just use the same Sequence object. If multiple rows in the same table need to share the same generated key value, they again can utilize one Sequence object. If multiple columns in a single table need generated values from DB2, then multiple column values can be generated from a single Sequence, or multiple Sequences can be used. Later in this article, you will see that a Sequence object can serve as a DB2 global variable and can even be used to generate alphanumeric key values.

Creating a DB2 Sequence

To create a DB2 Sequence, the new CREATE SEQUENCE SQL statement is used. On this statement, the programmer controls which numeric data type is used and stored by DB2 in

the sequence object. The data type can be any exact numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or NUMERIC) with a scale of zero. The following statement shows how to create a sequence object that generates integer employee ID values starting with a value of 1000 and incrementing by 10 to generate the next ID in the sequence.

**CREATE SEQUENCE employee_id AS INTEGER
START WITH 1000 INCREMENT BY 10**

On iSeries a Sequence object is created as a data area object, so when browsing objects in a library with the Display Library (DspLib) CL command the object type will be data area and not sequence. A Sequence should not be changed with the CL command Change Data Area (ChgDtaAra) or any other similar interface because doing so may cause unexpected failures or unexpected results when attempting to use the sequence through SQL. Despite the fact that a data area object is the foundation of a DB2 Sequence, there is no native support (e.g., CL commands) for creating and generating Sequence values. This is consistent with the strategy used in prior OS/400 releases with many new DB2 enhancements requiring the usage of SQL.

As Figure 1 shows, several Sequence attributes are available to define a key-generation behavior that best matches your business requirements. By default, DB2 UDB will start a Sequence object column with a value of 1 and increment that value by 1 each time a Sequence value is generated.

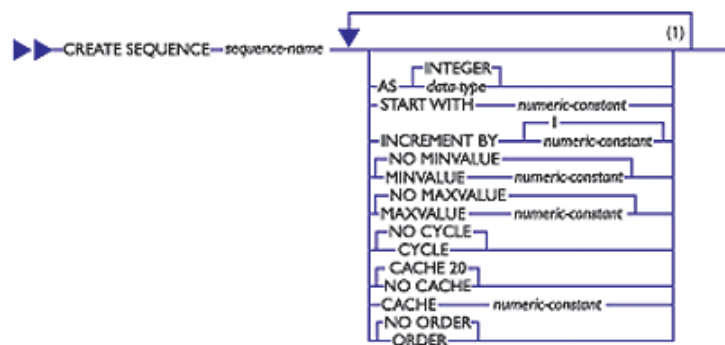


Figure 1
Syntax diagram for CREATE SEQUENCE

As the earlier example demonstrated, you can specify the START WITH and INCREMENT BY clauses to easily change this default behavior. The INCREMENT BY clause also lets you specify a negative value to generate a descending key value if needed.

The MINVALUE and MAXVALUE clauses allow a limited set of Sequence values to be generated. Once a sequence hits its minimum or maximum value it will either stop generating sequence values or cycle. The minimum and maximum values for a Sequence default to the min and max values associated with the data type of the Sequence.

When DB2 UDB hits the maximum or minimum value for a Sequence object, the default behavior of NO CYCLE prevents a new value from being generated. For cases in which you want DB2 UDB to continue assigning key values after the maximum or minimum value is

reached, you can specify the CYCLE attribute to have the Sequence start over at the beginning. For an ascending sequence, DB2 UDB would start assigning with the minimum value; for descending sequences, DB2 UDB would use the maximum value.

The Cache attribute specifies the maximum number of Sequence values that DB2 can pre-allocate the first time that a job or connection asks for a Sequence value to be generated. Caching a block of Sequence values within a job improves performance by reducing the number of times that DB2 needs to touch and lock the Sequence object to generate a new Sequence value. The default setting is "Cache 20" which means the first time that job asks for a sequence value to be generated, the next 20 sequence values will be cached within that job.

Figure 2a demonstrates the effect of the default Cache setting by showing how generated Sequence values are distributed between two jobs sharing the same Sequence object. The NEXT VALUE expression which is used to retrieve the next generated value from the Sequence object will be covered in detail later on.

```
CREATE SEQUENCE s1 CACHE 20 NO ORDER  
START WITH 1 INCREMENT BY 1  
Job1: NEXT VALUE FOR s1 => VALUE = 1  
Job 2: NEXT VALUE FOR s1 => VALUE = 21  
Job 1: NEXT VALUE FOR s1 => VALUE = 2  
Job 1: NEXT VALUE FOR s1 => VALUE = 3  
Job 2: NEXT VALUE FOR s1 => VALUE = 22
```

Figure 2A
Sequence behavior with caching

The first time that Job1 gets a sequence value the first 20 Sequence values (i.e., 1-20) are cached within that job. That's why the first time that Job2 gets a Sequence value it receives the value of 21 instead of 2 and the next set of 20 Sequence values (21-40) are cached within Job2. If a job ends before it uses all of the cached Sequence values, then those values are lost and will never be generated from the Sequence object again. These lost values mean that there can be gaps in the values generated from a Sequence. Gaps in the generated value may not meet the requirements of some applications. Specifying "NO CACHE" can prevent sequence values from being lost, but the tradeoff is slower performance because DB2 will have to access the Sequence object for every value generated in addition to locking the object on every access.

The previous example also shows the default Order setting of NO ORDER. This setting means that the sequence values need not be generated in order of request. That's exactly the behavior showed in Figure 2a, the second Sequence value retrieved is 21 instead of 2. If an application requires the values to be generated in order of request, then the ORDER attribute can be specified on the Sequence definition. Like the NO CACHE option, the Order

attribute should be used cautiously because it effectively disables the caching of values and forces DB2 to lock and touch the Sequence object each time that a Sequence value needs to be generated. Figure 2b takes the earlier example and demonstrates how the Order attribute changes the distribution of generated sequence values between the two jobs.

```
CREATE SEQUENCE s1 NO CACHE ORDER  
START WITH 1 INCREMENT BY 1  
Job1: NEXT VALUE FOR s1 => VALUE = 1  
Job 2: NEXT VALUE FOR s1 => VALUE = 2  
Job 1: NEXT VALUE FOR s1 => VALUE = 3  
Job 1: NEXT VALUE FOR s1 => VALUE = 4  
Job 2: NEXT VALUE FOR s1 => VALUE = 5
```

Figure 2B
Sequence behavior without caching

These different attributes can be used to create a Sequence object that returns a constant value. A constant sequence object gives you the ability of defining a global DB2 variable that can be shared across jobs or applications. A constant sequence can be created by specifying an INCREMENT value of zero and a START WITH value that does not exceed MAXVALUE, or by specifying the same value for START WITH, MINVALUE and MAXVALUE. For a constant sequence, each time a NEXT VALUE expression is processed, the same value is returned. The following example shows the definition of a constant sequence and then how the ALTER SEQUENCE statement can be used to change the value of a constant sequence being used as a global DB2 variable.

```
CREATE SEQUENCE global_variable AS INTEGER  
START WITH 10 INCREMENT BY 0 MAXVALUE 10  
...  
ALTER SEQUENCE global_variable RESTART WITH 20 MAXVALUE 20
```

Generating Sequence Values

Earlier examples show how the NEXT VALUE expression can be used to generate and receive generated values from a Sequence object. The NEXT VALUE (and PREVIOUS VALUE) expressions can be referenced almost anywhere that an SQL expression (e.g., concat(c1,c2) , col1+10) is allowed. The following example shows the NEXT VALUE expression being used with an Insert statement to generate an id for a new employee using the employee_id Sequence object.

```
INSERT INTO employee  
VALUES (NEXT VALUE FOR employee_id, 'BJ ARMSTRONG' )
```

When a value is generated for a sequence with the NEXT VALUE expression, that value is consumed, and the next time that a value is requested, a new value will be generated. This

is true even when the statement containing the NEXT VALUE expression fails or is rolled back.

If the generated employee id needs to be shared with other tables, than the PREVIOUS VALUE expression can be used as demonstrated below. In this example, the new row will contain the same employee id value that was used for the previous insert into the employee table.

```
INSERT INTO employee_jobhistory  
VALUES (PREVIOUS VALUE FOR employee_id, 'JUNIOR PROGRAMMER')
```

For each job, activation group or connection, DB2 remembers the most recently generated value for a Sequence, and returns this value for a PREVIOUS VALUE expression specifying the Sequence name. The previous value persists until either the next value is generated for the Sequence, the Sequence is dropped or altered, or the end of the job is reached. The value is unaffected by COMMIT and ROLLBACK statements.

Another method of sharing a generated sequence value across multiple tables is to store the results of the NEXT VALUE expression into a local host variable in your application program. This technique is demonstrated in the following examples.

```
VALUES NEXT VALUE FOR employee_id INTO :hostvar1
```

```
SELECT NEXT VALUE FOR employee_id INTO :hostvar2  
FROM sysibm.sysdummy1
```

Using the NEXT VALUE expression to store the generated sequence value into an application host variable provides a method for native programs to utilize Sequence objects. Instead of changing all existing native programs to use the SQL NEXT VALUE expression, a single program or module can be written that uses SQL to retrieve the generated Sequence value into a variable. Then, native programs can just call a single program or module to generate new key values and use these values on native write applications.

The ability to generate alphanumeric key values with Sequence objects was one of the superior capabilities over Identity columns discussed earlier. Alphanumeric key values are generated by casting (or converting) the result of a NEXT VALUE expression into a character string. This technique is used in the following example, a character string of 'N1001' will be inserted as the order number for this first order for 10 boxes of nails.

```
CREATE SEQUENCE order_id START WITH 1001 MAXVALUE 9999
```

```
INSERT INTO orders  
VALUE ('N'||CAST(NEXT VALUE FOR order_id AS CHAR(4))), 10, 'NAILS')
```

When casting a generated value to a character string, it's important to realize that the number of digits in the generated Sequence value should match the number of characters in the result string. If the number of digits is less, then the resultant character string could contain blanks. That is the reason that this example initializes the Sequence object with a numeric value with four significant digits.

Managing DB2 Sequences

The DB2 Global Variable example covered earlier showed the ALTER SEQUENCE statement in action. This statement is the only interface available for changing the attributes

of an existing Sequence object. Resetting a Sequence back to its original state is as simple as the following statement.

ALTER SEQUENCE s1 RESTART

This request just resets the Sequence object back to its original start value and leaves all of the other attributes unchanged.

If you're trying to determine if a DB2 Sequence object exists on your system or the attributes of an existing Sequence, then you can use iSeries Navigator to examine a Sequence. One could also query the SYSSEQUENCES catalog view in QSYS2

There are really no new management requirements for a DB2 Sequence object because they are implemented as OS/400 data area objects. A Sequence can be saved with a library or saved individually using the existing OS/400 Save and Restore commands.

Data areas can be journaled, so you can journal a Sequence object to improve the recovery and availability of applications dependent on a Sequence for generating key values. You can use the CL command Start Journal Object (STRJRNOBJ) to activate journaling for a DB2 Sequence.

Hopefully, you now have a good understanding of how DB2 sequence objects can be used to reduce the total amount of application code that you must write and maintain in support of key value generation. For more detailed information, consult the DB2 UDB for iSeries SQL Reference guide.