February 2004

DB2 update

Sequence Objects and Identity Columns

By Craig S. Mullins

When designing DB2 databases a frequently heard request is for a column that contains sequentially generated numbers. For example, each row has a counter associated with it. When a new row is inserted, the counter should be incremented by one for the new row. This way, each new DB2 row has a unique "row number" associated with it. Until recently such a design was difficult to deliver.

Without sequence objects or identity columns an application program can implement similar functionality, but usually not in a manner that performs adequately as database usage scales. A common technique is to maintain a one-row table that contains the sequence number. Each transaction locks that table, increments the number, and then commits the change to unlock the table. In this scenario only one transaction at a time can increment the sequence number. A variation uses something like this

```
SELECT MAX()+ 1
FROM   ONEROW_TABLE
WITH RR;
```

The result is the next highest number to be used. This value is used by the application and ONEROW_TABLE must be updated with the incremented value. Performance bottlenecks will occur with this method when a lot of concurrent usage is required.

But now DB2 offers two methods of automatically generating sequential numbers for a column:

- Identity columns, and;
- SEQUENCE objects.

Identity Columns

Identity columns were formally added to DB2 as of Version 7, but were actually available as of the DB2 Version 6 refresh. The identity property is applied to a DB2 column using the IDENTITY parameter. A column thusly defined will cause DB2 to automatically generate a sequential value for that column when a row is added to the table. For example, identity columns might be used to generate primary key values or a value that somewhat mimics Oracle's row number capability. Using identity columns helps to avoid some of the concurrency and performance problems that can occur when application programs are used to populate sequential values for a "counter" column.

When inserting data into a table that uses an identity column, the program or user will not provide a value for the identity column. Instead, DB2 automatically generates the appropriate value to be inserted.

Only one identity column can be defined per DB2 table. Additionally, the data type of the column must be SMALLINT, INTEGER, or DECIMAL with a zero scale, that is DECIMAL(*n*,0). The data type also can be a user-defined DISTINCT type based on one of these numeric data types. The designer has control over the starting point for the generated sequential values, and the number by which the count is incremented.

An example creating a table with an identity column follows:

```
CREATE TABLE EXAMPLE
   (ID_COL INTEGER NOT NULL
           GENERATED ALWAYS AS
IDENTITY
           START WITH 100
```

```
                INCREMENT BY 10
...);
```

In this example, the identity column is named ID_COL. The first value stored in the column will be 100 and subsequent INSERTs will add 10 to the last value. So the identity column values generated will be 100, 110, 120, 130, and so on.

Note, too, that each identity column has a property associated with it assigned using the GENERATED parameter. This parameter indicates how DB2 generates values for the column. You must specify GENERATED if the column is to be considered an identity column or the data type of the column is a ROWID. This means that DB2 must be permitted to generate values for all identity columns. There are two options for the GENERATED parameter: ALWAYS and BY DEFAULT.

- **GENERATED ALWAYS** indicates that DB2 will always generate a value for the column when a row is inserted into the table. You will usually specify ALWAYS for

your identity columns unless you are using data propagation.

- **GENERATED BY DEFAULT** indicates that DB2 will generate a value for the column when a row is inserted into the table unless a value is specified. So, if you want to be able to insert an explicit value into an identity column you must specify GENERATED BY DEFAULT.

Additionally, you can specify what to do when the maximum value is hit. Specifying the CYCLE keyword will cause DB2 to begin generating values from the minimum value all over again. Of course, this can cause duplicate values to be generated and should only be used when uniqueness is not a requirement.

Actually, the only way to ensure uniqueness of your identity columns is to create a unique index on the column. The IDENTITY property alone will not guarantee uniqueness.

Sometimes it is necessary to retrieve the value of an identity column immediately after it is inserted. For example, if you are using identity columns for primary key generation you may need to retrieve the value to provide the foreign key of a child table row that is to be inserted after the primary key is generated. DB2 provides the IDENTITY_VAL_LOCAL() function that can be used to retrieve the value of an identity column after insertion. For example, you can run the following statement immediately after the INSERT statement that sets the identity value:

```
VALUES IDENTITY_VAL_LOCAL() INTO
:IVAR;
```

The host variable IVAR will contain the value of the identity column.

Problems with Identity Columns

Identity columns can be useful, depending on your specific needs, but the problems that accompany identity column are numerous. Some of these problems include:

- Handling the loading of data into a table with an identity column defined as GENERATED BY DEFAULT. The next identity value stored by DB2 to be assigned may not be the correct value that should be generated. This can be especially troublesome in a testing environment.

- LOAD INTO PART x is not allowed if an identity column is part of the partitioning index.

- What about environments that require regular loading and reloading (REPLACE) for testing? The identity column will not necessarily hold the same values for the same rows from test to test.

- Prior to V8, it was not possible to change the GENERATED parameter (such as from GENERATED BY DEFAULT to GENERATED ALWAYS).

- The IDENTITY_VAL_LOCAL() function returns the value used for the last insert to the identity column. But it only works after a singleton INSERT. This means you cannot use INSERT INTO SELECT FROM or LOAD, if you need to rely on this function.

- When the maximum value is reached for the identity column, DB2 will cycle back to the beginning to begin reassigning values - which might not be the desired approach.

If you can live with these caveats, then identity columns might be useful to your applications. However, in general, these "problems" make identity columns a very niche solution. IBM has intentions to rectify some of these problems over time in upcoming versions of DB2.

SEQUENCE Objects

But remember, DB2 has two methods of automatically generating sequential numbers. The first method is to define an identity column for the table; the second is to create a SEQUENCE object. A SEQUENCE object is a separate structure that generates sequential numbers.

New to DB2 V8, a SEQUENCE is a database object specifically created to generate sequential values. So, a using a SEQUENCE object requires the creation of a database object; using an identity column does not.

A SEQUENCE objects is created using the CREATE SEQUENCE statement.

When the SEQUENCE object is created it can be used by applications to “grab” a next sequential value for use in a table. SEQUENCE objects are ideal for generating sequential, unique numeric key values. A sequence can be accessed and incremented by many applications concurrently without the hot spots and performance degradation

associated with other methods of generating sequential values.

Sequences are designed for efficiency and to be used by many users at the same time without causing performance problems. Multiple users can concurrently and efficiently access SEQUENCE objects because DB2 does not wait for a transaction to COMMIT before allowing the sequence to be incremented again by another transaction.

An example creating a SEQUENCE object follows:

```
CREATE SEQUENCE ACTNO_SEQ
   AS SMALLINT
   START WITH 1
   INCREMENT BY 1
   NOMAXVALUE
   NOCYCLE
   CACHE 10;
```

This creates the SEQUENCE object named ACTNO_SEQ. Now it can be used to generate a new sequential value, for example

```
INSERT INTO DSN8810.ACT
     (ACTNO, ACTKWD, ACTDESC)
   VALUES
     (NEXT VALUE FOR ACTNO_SEQ,
      'TEST', 'Test activity');
```

The NEXT VALUE FOR clause is known as a sequence expression. Coding the sequence expression causes DB2 to use the named SEQUENCE object to automatically generate the next value. You can use a sequence expression to request the previous value that was generated. For example

```
SELECT PREVIOUS VALUE FOR ACTNO_SEQ
INTO   :IVAR
FROM   DSN8810.ACT;
```

As you can see, sequence expressions are not limited to INSERT statements, but can be used in UPDATE and SELECT statements, too.

Caution: If you specify the NEXT VALUE FOR clause more than once in the same SQL statement DB2 will return the same value for each NEXT VALUE FOR specification.

SEQUENCE Object Parameters

Similar to identity columns, a SEQUENCE object has parameters to control the starting point for the generated sequential values, and the number by which the count is incremented. You can also specify the data type to be generated (the default is INTEGER). You can also specify a minimum value (MINVALUE) and a maximum value (MAXVALUE) if you wish to have further control over the values than is provided by the data type chosen.

Again, as with identity columns, you can specify how the SEQUENCE should handle running out of values when the maximum value is hit. Specifying the CYCLE keyword will cause the SEQUENCE object to wrap around and begin generating values from the minimum value all over again.

A final consideration for SEQUENCE objects is caching. Sequence values can be cached in memory to facilitate better performance. The

size of the cache specifies the number of sequence values that DB2 will pre-allocate in memory. In the previous example CACHE 10 indicates that ten sequence values will be generated and stored in memory for subsequent use. Of course, you can turn off caching by specifying NO CACHE. With caching turned off each new request for a sequence number will cause I/O to the DB2 Catalog (SYSIBM.SYSSEQUENCES) to generate the next sequential value.

SEQUENCE Object Guidelines

DB2 does not wait for an application that has incremented a sequence to commit before allowing the sequence to be incremented again by another application. Applications can use one sequence for many tables, or create multiple sequences for use of each table requiring generated key values. In either case, the applications control the relationship between the sequences and the tables.

The name of the SEQUENCE object indicates that we are going to use it to generate activity numbers (ACTNO), but its usage is not limited to that. Of course, failure to control the use of a SEQUENCE object can result in gaps in the sequential values. For example, if we use the ACTNO_SEQ object to generate a number for a different column, the next time we use it for ACTNO there will be a gap where we generated that number.

Other scenarios can cause gaps in a SEQUENCE, too. For example, issuing a ROLLBACK after acquiring a sequence number will not roll back the value of the sequence generator - so that value is lost. A DB2 failure can also cause gaps because cached sequence values will be lost.

Please note, too, that when sequences were introduced in non-mainframe DB2, syntax was supported that did not conform to the SQL standard. This non-standard syntax is supported on the mainframe as well:

- `NEXTVAL` can be used in place of `NEXT VALUE; and`
- `PREVVAL` can be used in place of `PREVIOUS VALUE`.

Choosing Between IDENTITY and SEQUENCE

Although both identity columns and SEQUENCE objects are useful for generating incremental numeric values, you will be confronted with situations where you will have to choose between the two. Consider the following criteria for when to use one instead of the other. Identity columns are useful when:

- Only one column in a table requires automatically generated values
- Each row requires a separate value
- An automatic generator is desired for a primary key of a table
- The LOAD utility is not used to load data into the table

- The process of generating a new value is tied closely to inserting into a table, regardless of how the insert happens

SEQUENCE objects are useful when:

- Values generated from one sequence are to be stored in more than one table
- More than one column per table requires automatically generated values (multiple values may be generated for each row using the same sequence or more than one sequence)
- The process of generating a new value is independent of any reference to a table

Unlike SEQUENCE objects, which are more flexible, identity columns must adhere to several rigid requirements. For example, an IDENTITY column is always defined on a single table and each table can have at most one IDENTITY column. Furthermore, when you create an IDENTITY column, the data type for that column must be numeric; not so for

sequences. If you used a SEQUENCE object to generate a value you could put that generated into a CHAR column, for example. Finally, when defining an IDENTITY column you cannot specify the `DEFAULT` clause and the column is implicitly defined as `NOT NULL`. Remember, DB2 automatically generates the IDENTITY column's value, so default values and nulls are not useful concepts.

Consult Table 1 for a summary comparison of sequences and identity column characteristics.

Table 1. Identity Columns Versus Sequence Objects.

Identity Columns	**Sequence Objects**
Internal objects generated and maintained by DB2	Standalone database objects created by a DBA
Associated with a single table	Not associated with a specific table; usable across tables
Use IDENTITY_VAL_LOCAL() to get last value assigned	Use PREVIOUS VALUE FOR *seq-expr* to get last value assigned
N/A	Use NEXT VALUE FOR *seq-expr* to get next value to be assigned
Add/change using ALTER TABLE ...ALTER COLUMN (*DB2 V8 only*) ...ALTER COLUMN (*DB2 V8 only*)	Administer using ALTER SEQUENCE, DROP, COMMENT, GRANT, and REVOKE
Version 6 refresh; Version 7	Version 8

Summary

Both identity columns and SEQUENCE objects can be used to automatically generate sequential values for DB2 columns. Prior to Version 8, identity columns are your only option. However, after you move to V8, SEQUENCE objects will provide more flexibility and be easier to use than the identity column option.

Happy sequential value generation with DB2!

From DB2 Update (Xephon) February 2004.

Home.