Thread D: **DB2 Universal Database for z/OS, V8**

Sequential Value Generators
in DB2 for z/OS

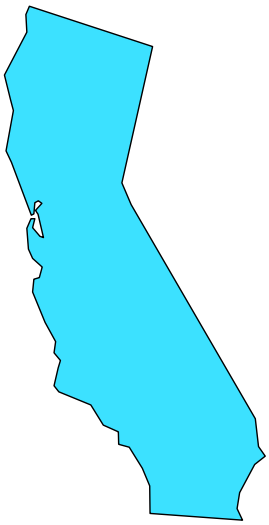
Ramani Croisettier
IBM Silicon Valley Lab

Session D8

Thursday, May 22nd 2003, 8.30AM

Sequential Key Value Generators in DB2 z/OS

Session D8



Ramani Croisettier

IBM Silicon Valley Lab

San Jose, California, USA

ramani@us.ibm.com

Agenda - page 1

- Function Overview
  - ★ Introduction
    - ▸ What are the Sequential Value Generators
      - ➟ Sequences, Identity Columns, Row IDs
  - ★ Benefits

- Function Details - Sequences
  - ★ SQL statements for Sequence Usage
  - ★ Defining a Sequence
  - ★ Sequence Attributes descriptions (several pages)
  - ★ Generating and Retrieving Sequence Values
  - ★ Altering Sequence attributes
  - ★ Using Sequences - Examples
  - ★ Altering Sequences - Examples

Agenda - page 2

- Function Details - Identity Columns
  - Defining Identity Columns
  - Identity Column attribute specifications
  - Required attributes description
  - Populating Identity Column Values
    - Identity Column Value while inserting a row
    - Identity Column Value while inserting with Subselect
    - Inserting row for Identity Column - Examples (3 pages)
    - Loading Row with Identity Column
  - Retrieving/Updating Identity Column Values
  - Altering Identity Column attributes
  - Adding Identity Column to non-empty table
  - Data Propagation with Identity Column tables
  - Clone Tables and Declared Temp Tables
  - List of V8 enhancements for Identity Columns

Agenda - page 3

- Usage Considerations
  - Catalog Tables and Value Generation
  - Performance considerations - Cache and Order
  - Logical Range of Values and Cycles
  - Uniqueness of values
  - Consumed Values
  - Use of Generated By Default attribute for Identity Column
  - Migration / Fallback
- Row IDs
  - How Sequences and Identity Columns differ from Row IDs

Agenda - page 4

- Error Situations
  - Exceeding Range of Values for Identity Columns
  - Incorrect use of Generated-By-Default
  - Are Gaps/Duplicates possible?
  - While RECOVERing Tablespaces ...
  - In case of System Crash
- Conclusion
  - Benefits of Identity Columns and Sequences
  - Future Directions

Introduction

- ★ What are Sequential Value Generators - Overview
  - ⋆ Sequence :
    - ▸ User-defined, stand-alone object not tied to a table
    - ▸ Generated values returned to user; can be used for any thing by user
  - ⋆ Identity Column :
    - ▸ Column with properties of a sequence; part of a table
    - ▸ DB2 Generated values directly applied to the column
  - ⋆ Row IDs
- ★ **Benefits**
  - ⋆ Problems with application solutions generating key values
    - ➙ Concurrency, Performance problems
    - ➙ Page contention, Retained lock in Data Sharing
  - ⋆ DB2 Benefits
    - ➙ High Concurrency, High Performance, Reliable, Recoverable
    - ➙ User-defined rules, DB2-managed
    - ➙ Easy to Define and Alter sequence properties

SQL Statements for Sequence Usage

- CREATE SEQUENCE <sequence-name> <attributes specification>
- ALTER SEQUENCE <sequence-name> <attributes specification>
- DROP SEQUENCE <sequence-name>
- COMMENT ON <sequence-name>
- GRANT <ALTER/USAGE> ON SEQUENCE <sequence-name>
- REVOKE <ALTER/USAGE> ON SEQUENCE <sequence-name>
- PREVIOUS VALUE FOR <sequence-name> expression
- NEXT VALUE FOR <sequence-name> expression

Defining a Sequence

- CREATE SEQUENCE <sequence-name>

(optional sequence attribute-specifications)

AS <data type>
START WITH <numeric value>
INCREMENT BY <numeric value>
NO MINVALUE / MINVALUE <numeric value>
NO MAXVALUE / MAXVALUE <numeric value>
NO CYCLE / CYCLE
NO CACHE / CACHE <integer value>
NO ORDER / ORDER
;

Example: CREATE SEQUENCE MySeq1 AS INTEGER
START WITH 1 INCREMENT BY 1
NO MINVALUE MAXVALUE 1000
CYCLE CACHE 20
ORDER;

*The **Data Type** attribute*

- Data Type
  - EXACT Numeric Data Type, or
    User-defined Distinct Type
    based on an EXACT Numeric Type
  - All user-specified values must be scale 0

- Supported exact numeric data types:
  - Small Integer (-32768 to +32767)
  - Integer (-2147483648 to +2147483647)
  - Decimal (1 to 31 digits precision, negative/positive)

- Default : Integer

*The **START WITH** attribute*

- First Value for the Sequence
  - START WITH keyword with value
  - Must be within range of data type, Scale 0
  - Can be outside of MinValue to MaxValue range
  - Example: Assume Decimal(2,0), Increment By = 1.
    Start With = 1, MinValue = 11, MaxValue = 99.
    (Generates values from 1 to 99 for 1st set.)

- Default
  - User-defined or default MINVALUE for ascending sequence
  - User-defined or default MAXVALUE for descending sequence

*The **INCREMENT BY** attribute*

- Interval between consecutive values
  - INCREMENT BY keyword with value
    - Within range of INTEGER data type, Scale 0
    - Can be 0
  - Determines Ascending or Descending Direction
    - Increment By < 0, Descending sequence
    - Increment By > 0, Ascending sequence
    - Increment By = 0
      - treated as Ascending sequence
      - could be used to generate a constant sequence
  - Default : 1

*The **MINVALUE** attribute*

- Minimum EndPoint of Range
  - MINVALUE keyword with value
    - Must be within range of data type, Scale 0
    - MinValue must be <= MaxValue
  - Ascending sequence cycles to MinValue

- Default (if NO MINVALUE specified either implicitly or explicitly)
  - For descending sequence,
    MinValue = minimum value of data type.
  - For ascending sequence,
    MinValue = Start With
    or 1 if Start With was NOT specified

*The **MAXVALUE** attribute*

- Maximum EndPoint of Range
  - MAXVALUE keyword with value
    - Must be within range of data type, Scale 0
    - MaxValue must be >= MinValue
  - Descending sequence cycles to MaxValue

- Default (if NO MAXVALUE specified either implicitly or explicitly)
  - For ascending sequence,
    MaxValue = maximum value of data type.
  - For descending sequence,
    MaxValue = Start With
    or -1 if Start With was NOT specified.

*The **CYCLE** attribute*

- CYCLE keyword: Wrap around and repeat
- NO CYCLE: Only one set of values, no repeating
- Default: NO CYCLE

- Repeating sequences
  - First cycle starts with START WITH value
  - Subsequent cycles start with
    - MinValue for ascending sequence
    - MaxValue for descending sequence

- Number of values in the 1st cycle may be different from number of values in subsequent cycles

*The **CACHE** attribute*

- Caching for better performance
  - CACHE keyword with integer value
    - minimum 2
  - NO CACHE keyword
- Default: CACHE 20
- Effect of caching in Data Sharing with NO ORDER
- CACHE and ORDER

*The **ORDER** attribute*

- ORDER / NO ORDER keywords
  - ORDER : Values to be assigned in order of request
  - NO ORDER
  - Default: NO ORDER
- In Data Sharing:
  - ORDER gets precedence over CACHE
  - Possible effect of NO ORDER
- In Non Data Sharing:
  - Keywords ORDER / NO ORDER have no effect over CACHE
  - Values always assigned in strict order

- In Data Sharing,
  If user specifies ORDER, then
  user-specified CACHE option is ignored and a default NO CACHE is used.
  If user specifies NO ORDER, then
  user-specified CACHE value or default CACHE 20 is used.

- In Non Data Sharing,
  ORDER and NO ORDER have no effect on CACHE.
  User-specified CACHE value or default CACHE 20 is used regardless of
  ORDER or NO ORDER.

Generating and Retrieving Sequence Values

★ NEXT VALUE FOR <*sequence-name*> *expression*
- Generates and returns the first/next value of sequence

★ PREVIOUS VALUE FOR <*sequence-name*> *expression*
- Returns the most recently generated value for the sequence for a previous statement within the current session
- NEXTVAL must have been invoked within current session

★ Restrictions on usage of the 2 expressions

ALTERing sequence attributes

- SQL statement:

```
ALTER  SEQUENCE  <sequence-name>
    INCREMENT BY   <numeric value>
    NO MINVALUE   /  MINVALUE   <numeric value>
    NO MAXVALUE  /  MAXVALUE  <numeric value>
    NO CYCLE   / CYCLE
    NO CACHE  / CACHE <integer value>
    NO ORDER / ORDER
    RESTART   /  RESTART WITH   <numeric value>
    ;
```

- At least 1 sequence attribue must be specified for the ALTER
- Data Type cannot be altered
- Unused cache values may be lost
- DB2 validation of ALTERed parameters ...

Using Sequences - Examples

```
CREATE TABLE T1 (C1 DECIMAL(31,0) NOT NULL, C2 DECIMAL(31,0) NOT NULL);
CREATE SEQUENCE S1 AS DECIMAL(31,0) START WITH 1, MINVALUE 1, CACHE 10;
INSERT INTO T1 (C1, C2) VALUES (NEXT VALUE FOR S1, NEXT VALUE FOR S1);
INSERT INTO T1 (C1, C2) VALUES (PREVIOUS VALUE FOR S1, NEXT VALUE FOR S1);
SELECT * FROM T1;
        |-----------------------------------------------|
        |          C1          |          C2            |
        |-----------------------------------------------|
      1 |             1        |            1           |
      2 |             1        |            2           |
        |-----------------------------------------------|
SELECT NEXT VALUE FOR S1 FROM SYSIBM.SYSDUMMY1;
          |-------------------------------------|
        1 |                            3        |
          |-------------------------------------|
SELECT NEXT VALUE FOR S1 FROM T1;
          |-------------------------------------|
        1 |                            4        |
        2 |                            5        |
          |-------------------------------------|
INSERT INTO T1 (C1, C2) VALUES (PREVIOUS VALUE FOR S1, NEXT VALUE FOR S1);
```

Altering Sequences - Examples

```
ALTER SEQUENCE S1 INCREMENT BY 100 RESTART;
COMMIT;
INSERT INTO T1 (C1, C2) VALUES (PREVIOUS VALUE FOR S1, NEXT VALUE FOR S1);
INSERT INTO T1 (C1, C2) VALUES (PREVIOUS VALUE FOR S1, NEXT VALUE FOR S1);
ALTER SEQUENCE S1 INCREMENT BY 1000 RESTART WITH 1000 CYCLE;
COMMIT;
INSERT INTO T1 (C1, C2) VALUES (PREVIOUS VALUE FOR S1, NEXT VALUE FOR S1);
INSERT INTO T1 (C1, C2) VALUES (PREVIOUS VALUE FOR S1, NEXT VALUE FOR S1);
SELECT * FROM T1;
```

	C1	C2
1	1	1
2	1	2
3	5	6
4	6	1
5	1	101
6	101	1000
7	1000	2000

Defining Identity Columns

SQL statements for defining Identity Columns:

* CREATE TABLE statement - example:

```
CREATE TABLE xTB1 (
    COLUMN1        Integer,
    IDENTITYCOL    < data type>   GENERATED . . .  as IDENTITY
                   < optional column attributes specifications >
  ) IN  xDB1.xTS1;
```

* ALTER TABLE ADD COLUMN statement - example

```
Alter Table  xTB2
Add IdentityCol2 < data type>  GENERATED . . .  as IDENTITY
                 < optional column attributes specifications >;
```

* ALTER TABLE xTB1 ALTER COLUMN IDENTITYCOL1
  < at least 1 optional column attributes specification >

* CREATE TABLE LIKE . . . statement with the
  Including Identity Attributes clause

Identity Column attributes specifications

- Data Type (required) :
  - same description as for Sequence definition

- Required Keywords : (described in the next page)
  - **GENERATED ALWAYS AS IDENTITY**
    OR
    GENERATED BY DEFAULT AS IDENTITY

- Optional Keywords : same description as for sequence definition
  - START WITH <numeric value>
  - INCREMENT BY <numeric value>
  - NO MINVALUE / MINVALUE <numeric value>
  - NO MAXVALUE / MAXVALUE <numeric value>
  - NO CYCLE / CYCLE
  - NO CACHE / CACHE <integer value>
  - NO ORDER / ORDER

Required keywords for Identity Column definition

★ GENERATED ALWAYS or GENERATED BY DEFAULT

Define source of Column Values

★ GENERATED ALWAYS :

➡ column can get only DB2-generated values

➡ user cannot supply value on insert

➡ user cannot update identity column

★ GENERATED BY DEFAULT

➡ column can get User-supplied and/or DB2-generated values

➡ Identity Column is updateable

★ AS IDENTITY

➡ Define column as Identity Column -
a Column with properties of a sequence

Populating Identity Column Values

- INSERT
- INSERT with SubSelect
- LOAD

Identity Column value while Inserting a row

- If GENERATED ALWAYS :
  - DB2 generates the column value per row. Value cannot be supplied
  - In INSERT statement
    - Don't list Identity column in column-list. Or
    - Specify DEFAULT in the values clause for identity column. Or
    - Use the OVERRIDING USER VALUE clause to tell DB2 to ignore the non-DEFAULT value specified for an Identity Column that is Generated Always.

- If GENERATED BY DEFAULT :
  - If user supplies column value, accepted
  - If column not in column-list, or user specifies DEFAULT for column value, then DB2 generates value
  - Cannot use the OVERRIDING USER VALUE clause

Identity Column value while Inserting with Subselect

- Same rules as for Insert

- If target column is Generated Always
  - Subselect cannot supply column value
  - Don't include identity column in column-list. Or
  - Specify DEFAULT in the VALUES clause for Identity Column. Or
  - Use OVERRIDING USER VALUE clause

- f target column is Generated By Default
  - Subselect can supply column value
  - If subselect does not provide column value then DB2 generates the value for the column.

Inserting row for Identity Column Generated Always Examples

```
CREATE TABLE TB1 (COL1  INTEGER NOT NULL,
                  IDCOL DECIMAL(15,0) GENERATED ALWAYS AS IDENTITY
                   (START WITH 10, INCREMENT BY 10, CACHE 10, ORDER) );

INSERT INTO TB1 (COL1) VALUES(1);
INSERT INTO TB1 (COL1, IDCOL) VALUES(2, DEFAULT);
INSERT INTO TB1 (COL1, IDCOL) OVERRIDING USER VALUE
                                  VALUES(3, 999);
SELECT * FROM TB1;
```

COL1	IDCOL
1	10
2	20
3	30

Inserting row with Subselect for Identity Column Generated Always - Examples

```
CREATE TABLE TB2 LIKE TB1
                    INCLUDING IDENTITY COLUMN ATTRIBUTES;
INSERT INTO TB2 SELECT * FROM TB1;
   SQLCODE = -798, ERROR: YOU CANNOT INSERT A VALUE INTO A COLUMN  THAT IS
    DEFINED WITH  THE OPTION GENERATED ALWAYS.  COLUMN NAME IDCOL.
INSERT INTO TB2 OVERRIDING USER VALUE SELECT * FROM TB1;
INSERT INTO TB2 (COL1) VALUES(4);
SELECT * FROM TB2;
```

COL1	IDCOL
1	10
2	20
3	30
4	40

Inserting row for Identity Column Generated By Default Examples

```
CREATE TABLE TB3 (COL1 INTEGER NOT NULL,
                  IDCOL DECIMAL(31,0) GENERATED BY DEFAULT AS IDENTITY);

INSERT INTO TB3 (COL1, IDCOL) VALUES(1, 1);
INSERT INTO TB3 (COL1)        VALUES(2);
INSERT INTO TB3 (COL1, IDCOL) VALUES(3, DEFAULT);
INSERT INTO TB3 (SELECT * FROM TB2 WHERE COL1 = 4);
SELECT * FROM TB3;
```

COL1	IDCOL
1	1
2	1
3	2
4	40

Loading Row with Identity Column

- LOAD Utility
- Generated By Default :
  - Can include Identity column in field-specification
  - Can use DEFAULTIF clause. If condition is met, column value will be generated by DB2
- Generated Always :
  - Column cannot be included in field-specification
  - Column cannot be implied by LOAD FORMAT UNLOAD or LOAD with no field list

Retrieving Identity Column values

- Normal 'SELECT' of column value
- Insert with Select
- Identity_Val_local() function invocation
  - Restricted to Agent scope
  - Nondeterministic function that returns the most recently assigned value for an identity column of any table by the same (invoking) application process within current Agent-scope
  - Invocation Examples:
    - SELECT IDENTITY_VAL_LOCAL() FROM SYSIBM.SYSDUMMY1;
    - INSERT INTO T1 (C2) VALUES (IDENTITY_VAL_LOCAL());

Updating Identity Column Values

- UPDATE assignment statement
- Update allowed for
  GENERATED BY DEFAULT column values
  - No validation beyond data type
- Update not allowed for
  GENERATED ALWAYS column values

ALTERing Identity Column Attributes

- SQL statement

```
ALTER TABLE  < table-name > ALTER COLUMN < identity-column-name >
      SET GENERATED ALWAYS  /  SET GENERATED BY DEFAULT
      SET INCREMENT BY                    <numeric value>
      SET NO MINVALUE   /  MINVALUE   <numeric value>
      SET NO MAXVALUE  /  MAXVALUE  <numeric value>
      SET NO CYCLE   / CYCLE
      SET NO CACHE  / CACHE               <integer value>
      SET NO ORDER / ORDER
      RESTART   /  RESTART WITH           <numeric value>
  ;
```

- At least 1 sequence attribue must be specified for the ALTER
- Data Type cannot be altered
- Unused cache values may be lost
- DB2 validation of ALTERed parameters ...

Adding Identity Column to a non-empty table

- ★ Alter Table <table name>Add column <identity column name>
  - Tablespace enters REORP (Reorg Pending) state
  - Tablespace must be REORGed before other use
    - ➡ Identity column values generated for existing rows
- ★ Alter Table Add column to empty table
  - No REORP state

Data Propagation

- Copying contents of one table to another
- Loading rows into Identity Column Tables
  - Use GENERATED BY DEFAULT for target Identity column
  - Source column need not be Identity, but data type must be compatible
  - Create index on Identity Column for uniqueness
  - Use appropriate START WITH value for DB2 to continue loaded sequence

Clone tables and Declared Temp Tables

- Creating a table "Like" another table containing Identity Column
  - Use the INCLUDING IDENTITY ATTRIBUTES clause to inherit identity column properties
  - Otherwise no inheritance
- Declared Temp Tables
  - Can contain Identity Column
  - SysIbm.SysSequences not updated

List of version-8 Identity Column Enhancements

- Allow altering attributes of identity column
  - ALTER TABLE ALTER COLUMN statement extended
- ORDER gets precedence over CACHE in Data Sharing
- **LOAD** enhancements
- Allow Increment By to be 0
- Allow MinValue = MaxValue
- Allow NO MINVALUE, NO MAXVALUE, ORDER, NO ORDER keywords
- Some minor sqlcode changes

Catalog Tables and Value Generation

- ★ Catalog Tables
  - ● SYSIBM.SYSSEQ tablespace
    - ‣ SYSIBM.SYSSEQUENCES table
      - ➡ One row per Identity Column or Sequence
      - ➡ Records all data about sequence attributes
      - ➡ MAXASSIGNEDVAL column
  - ● SYSIBM.SYSSEQ2 tablespace
    - ‣ SYSIBM.SYSSEQUENCESDEP table
    - ‣ SYSIBM.SYSSEQUENCEAUTH table
- ★ Value generation flow
  - ‣ Driven by Next Value expression or Insert for Identity Column
  - ‣ Update of MaxAssignedVal
  - ‣ Logging of Catalog Update
- ★ In case of system crash, as long as no cold start is done,
  - ‣ Still guaranteed unique, possible gaps
  - ‣ Surviving members in DS not affected
  - ‣ No retained locks to prevent number generation on surviving members

Performance Considerations - Cache and Order

- Concurrency in Data-Sharing
- Caching for Performance
  - Caching and Catalog I/O frequency
- Cache and Order : Effect of Caching In
  - Data Sharing environment
    - Concurrent use of same identity column or Sequence by applications on multiple DB2 members
      - Order of Requests and Order of Values
      - Precedence of ORDER over CACHE
  - Non-Data Sharing
      - Values assigned always in order of Requests
      - Order / No Order have no effect on CACHE

Logical Range of Values

- Logical Range of Values : Determined by
  - Data Type,
    MINVALUE, MAXVALUE, INCREMENT BY and START WITH

  - Example: Decimal(1,0) sequence allows range -9 to 9. But,
    - MinValue = 1, Start With 1, Increment By = 3
      narrow range to (1 to 7).

    - If Start With is not the same as MinValue for ascending sequence,
      Or Start With is not the same as MaxValue for descending sequence,
      then
      Logical range of values for first cycle is different from
      Logical range of values for subsequent cycles.

Cycles

Range of 1st set of values could be different from range of subsequent sets.
Example: Assume a Decimal(1,0) sequence which allows range -9 to 9.
Table below shows the 1st set of values and the subsequent sets of values for sequences with different attributes:

- Note: Sequence S4 is a descending sequence.

	Sequence S1 --- IncrementBy = 1, StartWith = 1, MinValue = 5, MaxValue = 9	Sequence S2 --- IncrementBy = 1, StartWith = 5, MinValue = 1, MaxValue = 9	Sequence S3 --- IncrementBy = 1, StartWith = 9, MinValue = 3, MaxValue = 8	Sequence S4 --- IncrementBy = -1, StartWith = 9, MinValue = 1 MaxValue = 5
Cycle 1	1 2 3 4 5 6 7 8 9	5 6 7 8 9	9	1 2 3 4 5 6 7 8 9
Cycle 2	5 6 7 8 9	1 2 3 4 5 6 7 8 9	3 4 5 6 7 8	1 2 3 4 5
Cycle n	5 6 7 8 9	1 2 3 4 5 6 7 8 9	3 4 5 6 7 8	1 2 3 4 5

Uniqueness of values

- Uniqueness of DB2 generated values guaranteed per CYCLE
  - For Sequence objects
  - For Identity Columns that are Generated Always, as long as the Generated Always attribute is retained

- For Identity Columns that are Generated By Default,
  - Uniqueness guaranteed only among DB2-generated values
  - Define unique index to ensure uniqueness
  - Unique index recommended but not enforced by DB2

Consumed Values

- Consumed Values
  - Consumed values not reused in current cycle
  - Assigned but not utilized values
  - Allocated but not assigned cache values

Use of Generated By Default attribute

- Use Generated By Default only Identity Columns only for data propagation or unload/reload
- If you mix DB2-generated and User-supplied values, be aware:
  - MaxAssignedVal not updated for inserts with user-supplied values; SysSequences gets out of synch with user table
  - Attributes specification applies to DB2-generated values
  - User-supplied values
    - not validated beyond data type
    - could mess up intended sequence
    - could be out of 'logical' range
    - could cause duplicates if no index exists, and Unique-key violation otherwise
    - could introduce gaps in sequence . . .

Migration to DB2 V8

- Sequences not available in Compatibility mode
- Available only in V8 New Function Mode

Row IDs

★ How Sequences and Identity Columns differ from Row IDs

Error Situations

- Exceeding range of values
- Incorrect use of Generated-By-Default
- Are Gaps/Duplicates possible?
- While RECOVERing Tablespaces ...
- In case of System Crash
  Identity-columns and Sequences may be affected by
  - Cold Start
  - Conditional Restart skipping forward recovery
  - Disabling data sharing
  - Data inconsistency can result for any object with cold start or when recovery phase is skipped, preventing log processing in recovery stage

Conclusion

★ Benefits of Identity Columns and Stand-alone Sequences
- Automatic sequential key-value generators
- High Performance
- Concurrency in Data Sharing Group
- Reliability: Guaranteed unique values (within Cycle) within DB2 and among Data Sharing Members
- Recoverability
- Powerful dynamic ALTER capability

★ Future Directions

Sequential Value Generators in DB2 for z/OS
Session D8

Ramani Croisettier
IBM Silicon Valley Lab

ramani@us.ibm.com