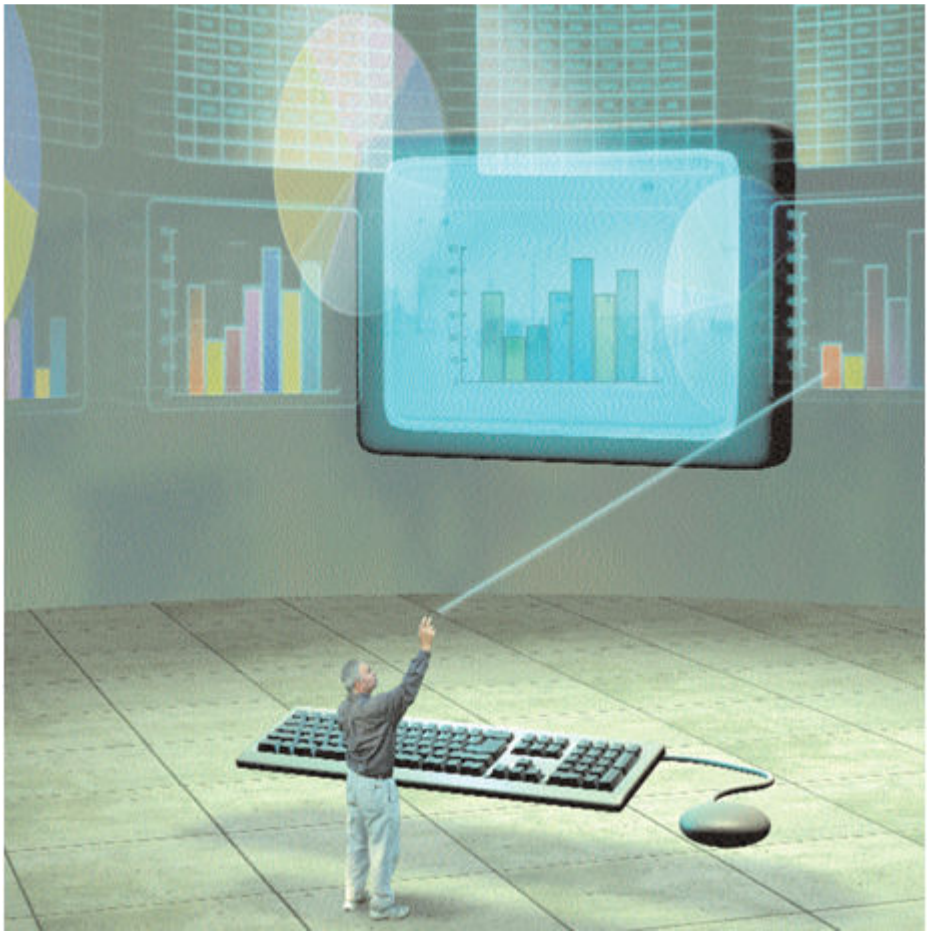
Data Orchestration

V5R3 enhancements improve DB2 UDB's ease of use

By Robert Andrews



Do you want to encrypt DB2* Universal Database* (UDB) data on your iSeries* server? Have you ever wished data could be auto-incremented in a table? Have you ever wanted to reorganize a table while your production applications are still active? Do you desire easier integration of ILE RPG and SQL? IBM* DB2 UDB for iSeries V5R3 is helping make these dreams come true. V5R3 offers several enhancements requested by users that help simplify the usage and programming experience. This article details these enhancements and more.

Sequence Objects

New SQL objects, called sequence objects, have been added that allow numeric values to be automatically generated. While this is similar to identity columns introduced in V5R2, sequence objects can be shared across multiple SQL objects. They can be defined for any numeric data type that has a

scale of zero, including user-defined types. On the i5/OS* side, they appear as data area objects. However, they have a signature attached that invalidates the sequence object if its altered from a non-SQL interface.

When defining a sequence object, specify the data type, starting value, increment value, minimum value, maximum value, cycling, caching and ordering. The starting value can be positive or negative. The incremental value can be any whole number to tell how many steps from the current to the next value. If this value is zero, a constant value is generated. If the value is negative, the sequence goes in descending, rather than ascending, order. When the sequence object hits the minimum or maximum value, it either cycles or stops generating values depending on how its set up.

To improve performance, an application may cache a set of values to be used. While this may enhance performance, it may also cause the sequence values to not be generated in order. In these cases, ordering can be forced. Assume we have a sequence with a cache of 20 with ordering, a starting value of 1 and an increase value of 1. If two jobs are going to be used to insert records-job A inserting a row, then job B, and job A again-the values would be 1, 2 and 3. However, if the same sequence object was created without ordering, the values would be 1, 21 and 2.

Sequence objects use the NEXT VALUE clause. For example, assuming a sequence S1 exists, to insert the next value into the table, use:

INSERT INTO TABLE1(NUM1, CHAR1) VALUES(NEXT VALUE FOR S1, Text value)

If NEXT VALUE is used multiple times in the same SQL statement, the same value will be returned for each reference to NEXT VALUE in the statement. The value last generated within that application process can be retrieved with the PREVIOUS VALUE clause. This can be useful when multiple tables must be updated as the first table generates the NEXT VALUE, and then the additional tables use the PREVIOUS VALUE clause. All tables then have the same value. Sequence objects can be used to generate non-numeric values by concatenating the sequence value with a character such as:

CONCAT(N, CAST(NEXT VALUE FOR S1 AS CHAR(4)))

Sequence objects can also be altered, dropped, labeled, commented and authority granted. When altering a sequence, almost all attributes can be changed including data type, next value, increment, minimum and maximum values. Cycling, caching and ordering can be altered as well. When a sequence is altered, all currently cached values for applications are discarded.

SQL Column Encryption

With increasing data-privacy concerns, customers are more interested in encrypting the data on their systems. DB2 UDB V5R3 provides built-in encrypt and decrypt SQL functions so native programs arent able to use the encrypted data without using embedded SQL or other SQL items (e.g., SQL views with stored passwords or SQL triggers) to handle the encryption and decryption.

Also with V5R3, encryption is handled at the field level. This means that the encryption and decryption occur when a particular field of a particular row is accessed. Theres no way at the file level to mark the table as "encrypted." This means that each field of each row can have its own password, which must be used to encrypt and decrypt the data.

The SQL scalar function, ENCRYPT_RC2, is used for encryption via the RC2 encryption algorithm. This function takes three parameters-the data to encrypt, the password and an optional hint. If the data value "123-45-6789" was encrypted with a password of "IBM" and a hint into a field called "SSN", the statement would look like:

INSERT INTO TABLE1(SSN) VALUES(ENCRYPT_RC2(123-45-6789, IBM, My Company Name))

To encrypt data, the fields length is longer than plain-text length as it contains an encryption header as well as the encrypted data. Assuming a single-byte character set and no binary large objects (BLOBs) are involved, the length is equal to the length of the plain-text data, plus the number of bytes to the next 8-byte boundary of the plain-text data, plus 8 bytes for the encryption header. So, to encrypt a string of "ABCDEF," the length of the result is 6 (the length of the plain text) plus 2 (number of bytes to the next 8-byte boundary) plus 8 (encryption header). So "ABCDEF" would need a 16-byte field once encrypted. If a hint is provided, an additional 32 bytes are required regardless of the hints length. For calculations involving BLOBs or double-byte character sets, see the SQL Reference in the iSeries Information Center.

Fields containing encrypted data must be declared as CHAR(n) FOR BIT DATA. If not, the data may be promoted to for bit data, which allows the encryption to work but not the decryption since the field type isn’t for bit data. To decrypt data, the result data type determines what function is used. The decrypt functions are DECRYPT_CHAR, DECRYPT_BINARY, DECRYPT_BIT and DECRYPT_DB (double byte). The function takes three parameters-the encrypted data, the password and, optionally, what coded character set identifier the data should be returned as. To get the aforementioned encrypt example back to plain text, use the statement:

SELECT DECRYPT_CHAR(SSN, IBM) FROM TABLE1

The results length equals the length of the original, plain-text data. Keep in mind that each field of each row can have its own password, so selecting all rows from the table may not work. Hence, the WHERE clause must be used to limit the rows selected for decryption to those that use the password provided in that statement.

Finally, the GETHINT function takes in the encrypted data and returns the hint stored with that field. Again, using the aforementioned example would return "My Company Name":

SELECT GETHINT(SSN) FROM TABLE1

When using Distributed Relational Database Architecture* to access encrypted data from a remote system, keep in mind that the password and returned data are sent as plain text. To help ensure security, IPSec or SSL should be used to encrypt communications between the two systems. For additional details and examples, see, "The Next Step in Security".

Parallel and Online Reorganize

To improve file efficiency, deleted records can be compressed by reorganizing the physical file member. Prior to V5R3, the file had to be taken offline to be reorganized and no other process could access the file during that time. A duplicate copy was created that required twice the amount of disk space. Logical files were then rebuilt. During this entire process, the file was unavailable.

With V5R3, online or cancelable reorganize was introduced. This allows a file to be reorganized while its being used. The file is journaled while the reorganize occurs and rows are moved one at a time. While this requires space for journal entries, the journal receiver threshold controls the disk usage with system-managed journals and by deleting receivers. The logical files over this physical file can be maintained with each row move or rebuilt at the end. Since the process involves moving records at the row level, if the DB2 Symmetric Multiprocessing (SMP) option is installed, the reorganize can be done in parallel. The access level that other users have to the file while its being reorganized can also be set by the reorganize command. If a reorganize is paused or suspended and then later resumed, it may pick up from where it left off, or, if significant changes were made to the file, it may start over again. For details on this feature, see, "DB2 UDB Gets Organized."

Journaling

Journaling goes hand-in-hand with DB2 for iSeries. Journaling with V5R3 features a few enhancements:

- A new receiver size option, *MAXOPT3, allows the sequence number of journal entries to go to 18,446,744,073,709,551,600.
- Several defaults have also been changed. While they wont affect journals that are migrated to V5R3, they will affect newly created journals. A default threshold of 1.5 GB instead of *NONE is set for journal receivers, and the journals are set to system-managed receivers instead of user-managed.
- The new save while active with partial transactions allows saves to be done mid-transaction without requiring the application to get to a commit boundary to save. For details on this new functionality, see the Backup and Recovery section in the iSeries Information Center.
- Remote journaling now has the capability to detect when its falling behind in sending entries to a remote system. Multiple rows can be bundled and broken on non-record boundaries increasing communication efficiency, which is known as super bundling. But theres no external screen that shows if the regular or super-bundling mode is currently being used. (Note: This is different from *SYNCPEND or *ASYNCPEND, which indicates catch-up mode is active. Catch-up mode is used when a remote journal is first activated to move the set of receivers from the source to the remote system.)

SQL Functions

Several enhancements to SQL functions and statements were also made at V5R3:

- When joining two tables that use common column names that must be matched, instead of using:

  FROM T1 JOIN T2 ON T1.C1 = T2.C1 AND T1.C2 = T2.C2 < /SPAN >

  the USING keyword can be substituted:

  FROM T1 JOIN T2 USING (C1, C2)

- Similar to the UNION clause, the new INTERSECT statement returns all rows that are in both of the tables intersected together. In contrast to INTERSECT, the EXCEPT statement returns all rows that are in the first result set but not in the second.
- A new RIGHT function works the same as the LEFT function but returns data from the right end instead of the left. Using RIGHT(ABCXYZ, 3) would return "XYZ."
- Using the REPLACE function, a string is searched for and replaced with another string.
- Using the INSERT function, an offset in the string is replaced with another. For example:

  REPLACE(ABCXYZ, ABC, DEF)

  returns "DEFXYZ", and

  INSERT(ABCXYZ, 1, 3, DEF)

  also returns "DEFXYZ"-the first searching for "ABC," and the second replacing characters starting at offset one for three characters.

- The REPEAT function repeats a string for a set number of times to produce one long string: REPEAT(ABC, 3) returns "ABCABCABC."
- Multiple rows can now be added with one INSERT statement. For example, if TABLE1 had a numeric and character string, such as:

  INSERT INTO TABLE1 VALUES (1, ABC), (2, DEF)

  it would add two rows, one with "1" and "ABC" and the other with "2" and "DEF." Now this syntax also allows SQL stored procedures to do blocked inserts.

- External SQL stored procedures point to a high-level language (HLL) program to run. Starting in V5R3, stored procedures can point to an entry point in a service program. When creating the stored procedure, the external name parameter would be LIB1/SRVPGM1(PGM1) where "SRVPGM1" is the service program and "PGM1" is its entry point. (Note: The entry point is case-sensitive.)

SQL ILE RPG Precompiler Enhancements

While ILE RPG programmers have wanted to embrace SQL in ILE RPG, many have felt limited by the precompilers restrictions. In past releases, the SQL ILE RPG Precompiler, which allows SQL statements to

be embedded directly into an ILE RPG program, allowed only a small subset of ILE RPG Compiler functions. In V5R3, the precompiler was enhanced to include many features that native ILE RPG programmers have been waiting for.

The first of these enhancements includes being able to use qualified subfields. The precompiler scans for the QUALIFIED keyword that allows the same subfield name in multiple data structures. In addition to qualified data structures, array data structures can now be used. Array data structures are declared with the DIM(n) keyword. At this time, array data structures can only be used for blocked fetches or blocked inserts since array-index references still arent allowed. If the data structure isnt an array, the LIKEDS keyword can be used to define one data structure based on a previously defined one. Likewise, the LIKEREC keyword defines a data structure based on an externally described file format. The precompiler doesnt support the optional second parameter of LIKEREC.

The last major enhancement is the preprocessor option. The precompiler calls the preprocessor to expand the source and allow the use of compiler directives. In the past, only single-level /COPY statements were processed. Now, depending on the value set for the RPG preprocessor options (RPGPPOPT) parameter, several compiler directives can be used-/IF, /ELSE, /ELSEIF, /ENDIF, /EOF, /DEFINE and /UNDEFINE.

To not use the RPG preprocessor, the value can be set to *NONE. The precompiler still picks up and uses single-level /COPY statements. A value of *LVL1 calls the RPG preprocessor and expands all /COPY commands, including nested /COPY statements and conditional-compile directives. In addition to the functions in *LVL1, *LVL2 expands /INCLUDE directives. Using *LVL1 or *LVL2 increases the possibility that the expanded source generated by the RPG preprocessor will become very large and a resource limit may be reached, depending on the nesting levels and expansion performed. If a limit is reached, the source must be broken into smaller pieces or the preprocessor cant be used.

Offering Easier Data Access

Overall, DB2 UDB for iSeries offers many enhancements in usage and programming methods to allow easier data access. Sequence objects provide more flexibility and expand on identity columns. Data encryption makes sensitive information more secure. Additional SQL functions provide more data selection and string manipulation. The ILE RPG precompiler and preprocessor options allow for easier integration of ILE RPG and SQL. For details on the exact parameters and values to be used in these commands, see the DB2 UDB for iSeries SQL Reference, SQL Programming, Embedded SQL Programming, and the Backup and Recovery manuals, which are available in the iSeries Information Center.