Manage SQL Entities within MDCMS

from Midrange Dynamics

From Version 8.5

Published May 10, 2023

Attribute Settings for SQL Entities

MDCMS provides the necessary attribute types and deployment features to be able to easily and consistently manage all SQL for DB2 entities on the IBM i.

Attribute Types

Entity	MDCMS SQL Type
Alias	*SQLALS
Constraint	*SQLCST
Function	*SQLFUN
Index	*SQLIDX
Materialized Query Table	*SQLMQT
Procedure	*SQLPRC
Data Manipulation Script	*SQLSCR
Sequence	*SQLSEQ
Table	*SQLTAB
Trigger	*SQLTRG

Entity	MDCMS SQL Type
User Defined Type	*SQLUDT
User Defined Global Variable	*SQLVAR
View	*SQLVW

All SQL Type attributes can be easily generated with the appropriate properties and create commands within MDCMS by pressing F9=Gen Dft Attr from within the Attributes option of the MDCMS Setup Menu.

Object Library

The Object Library parameter for the attribute allows for the system name or SQL long name of the target library (schema) for the attribute. Due to runtime performance, we highly recommend setting the Object Library value as follows:

- If the attribute type is *SQLFUN or *SQLPRC, the Object Library name should be the SQL long name for the schema.
- If the attribute type is NOT *SQLFUN or *SQLPRC, the Object Library name should be the 10-character system name for the library.

If using Object Replication templates, the list of libraries to replicate to is always by system name.

Source Library/File

It is recommended to always migrate source from level to level and then run a creation script to create the entity into the object library. This is mandatory for any entity that doesn't explicitly exist as a system object.

SQL scripts can reside as a member in a source file or as a file in the IFS.

When as a member, specify the target source file and system name of the source library for the member on the attribute.

When as a file in the IFS, specify source file of *IFS and the full path from the root for the source library, such as /dev/source/sql

If not permitted to retain source on a target partition, such as production, set the source library value to *TEMP.

Dft Source Naming

Since the names of SQL entities can be quite long, we recommend keeping the source in the IFS and then setting the Dft Source Naming parameter to ++OBJNAM++.sql. Then, when the entitiy is checked out, MDCMS will automatically assume that the IFS file is the same as the SQL long name with file type .sql.

If your source will be stored in source members, then we recommend using identifiable abbreviated names for the source members that match explicitly defined system names for the objects (where applicable) which are typically set with the FOR SYSTEM NAME clause in the SQL script. This makes it simpler for others to match the source to the object.

Attribute Commands

It is recommended to use the RUNSQLSTM command to generate the SQL entity.

The command type will be C=Compile, unless the attribute type is *SQLCST, *SQLSCR or *SQLTRG, in which case the command type will be 3=Post-Installation.

The default recommended syntax of the attribute command with source in a member is:

RUNSQLSTM SRCFILE(##SRCLIB##/##SRCFIL##) SRCMBR(##SRCNAM##) COMMIT(*NONE)

NAMING(*SYS)

The syntax of the attribute command with source in IFS is:

RUNSQLSTM SRCSTMF('##SRCLIB##/##SRCNAM##') COMMIT(*NONE) NAMING(*SYS)

The *SYS naming parameter should NOT be changed to *SQL. *SYS still uses SQL naming but uses the library list for locating unqualified entities, which is necessary at creation time, since MDCMS first creates the entities into a temporary library.

If the MARGINS parameter is omitted, MDCMS will automatically set the MARGINS value based on the length of the target source file and add 40 characters to it to allow for wildcard values.

IMPORTANT: Ensure that the Wildcards in SQL parameter on the MDCMS command definition is set to Y (true). Even if you don’t have any wildcards in the script, MDCMS will then properly qualify the entity to be created and ensure the margin values, etc. get set appropriately. If Wildcards in SQL isn’t set to Y, there is a high risk for certain entity types, or if using the OR REPLACE clause, that the entity will get created in the wrong library.

SQL Script Syntax

The creation scripts should not use the DROP statement, since MDCMS manages this automatically and can then reinstate the entity in the case of a rollback.

The creation scripts may use the OR REPLACE clause, but it is mostly not applicable because MDCMS first automatically DROPS the entity before creating it. The exception to this is for SQL variables (*SQLVAR) where MDCMS doesn’t perform a DROP due to the risk of dropping dependencies.

If using the OR REPLACE clause, it is vitally important that the Wildcards in SQL parameter on the MDCMS command definition is set to Y, otherwise, the entity may get created in the wrong library.

The ALTER statement, in lieu of a CREATE statement, should largely be avoided, for the reason of rollback capabilities and to take advantage of minimal downtime when using MDRapid. If an ALTER rather than CREATE must occur, then it should be part of a U=Update command type rather than C=Compile command type.

The exception to ALTER usage is in the case of Constraints, since they are added or changed as a post-installation command using an ALTER TABLE statement. For the other entities, the first executable statement in the script should be one of the following based on the type:

`CREATE ALIAS`

`CREATE FUNCTION`

`CREATE INDEX`

`CREATE PROCEDURE`

`CREATE SEQUENCE`

`CREATE TABLE`

`CREATE TRIGGER`

`CREATE TYPE`

`CREATE OR REPLACE VARIABLE`

`CREATE VIEW`

Library names should never be hardcoded in the CREATE script. Wildcards aren't necessary either and should rarely be used for DB objects that the entity references, unless they aren't located in the intended location in the library list for the target environment. A custom wildcard is recommended in this case to point to the correct library that the entity references for the environment.

The ++OBJLIB++ wildcard is permitted to qualify the entity that is to be created, but isn't required. MDCMS automatically temporarily sets the current library to the system name of the target schema prior to executing the script which causes the entity to be placed in that library.

This isn't the case for SQL indexes, views, constraints or triggers – when unqualified, they are placed in the same schema as the first table they reference or the table attached to in the case of the SQL constraint or trigger. To avoid this situation, it is important to specify Y for command parameter Wildcards in SQL and then MDCMS will automatically qualify the entity prior to executing the script even if a wildcard isn't present. This means that the SQL script doesn't ever need wildcards in it so that portability of the script is enabled. For constraints only, the table is being qualified and this requires the table to be in the same schema as the constraint.

The creation scripts should not use the RENAME statement to rename an SQL index, table or view SQL long name or system name. Rather the creation scripts should use the FOR SYSTEM NAME clause to control the name of the system name assigned to an SQL long named object.

The creation scripts should not use the GRANT statement to assign authority to an object. Rather the IBM i object system authorities should be used instead. MDCMS manages object authority through object authority templates. If the GRANT statement is used, the authority will still be replaced with the values of the object authority templates.

Requesting an SQL Object

When creating an object request for an SQL attribute, it is recommended to use the SQL long name as the object name. In the case of SQL indexes, sequences, tables and views, the 10-character system name can be used as the object name, if preferred.

MDCMS v8.3 and lower accepts SQL long names up to a length of 80 characters and MDCMS v8.3.1 and higher accepts SQL long names up to a length of 128 characters.

If requesting a long name from the MDCMS Object Manager in a 5250 session, press F2 while the cursor is on the object name field to be able to enter the entire long name.

IMPORTANT: The value of the object name on the object request MUST match the long name or system name of the object in the SQL script. After the script is run, MDCMS checks the system catalog to see if the name of the object on the object request now exists in the target library for the given entity type.

If you run the Create Object option or run the RFP and MDCMS returns an error even though the script finished without errors, it is almost always because the name of the object on the object request doesn't match the name of the object in the script.

For Functions and Procedures, the name of the Object on the object request must match the SPECIFIC name of the routine. This is because there can be duplicate routine names, but only one specific name.

SQL Entity Details

*SQLALS – Alias

Description

An alias is an alternate name for a table or view.

Sytem Type

*FILE

System Catalog Table

QSYS2/SYSTABLES where TABLE_TYPE = ‘A’

Special Handling in MDCMS

*SQLALS attributes contain the following 2 additional parameters

- For Database, represented by wildcard ++RMTRDB++
- For Library, represented by wildcard ++RMTLIB++

These wildcards can then be used in the SQL script to point to the appropriate database and library for the table/view that the alias is over, in order to be able to have a single script for all environments for an alias.

Example:

CREATE ALIAS MY_ALIAS

FOR ++RMTRDB++.++RMTLIB++.MY_TABLE;

*SQLCST – Constraint

Description

A constraint is a rule that the database manager enforces.

There are three types of constraints:

- A unique constraint is a rule that forbids duplicate values in one or more columns within a table. Unique and primary keys are the supported unique constraints.
- A referential constraint is a logical rule about values in one or more columns in one or more tables.
- A check constraint sets restrictions on data added to a specific table.

Sytem Type

none

System Catalog Table

QSYS2/ SYSCST

Special Handling in MDCMS

It is recommended that check, primary key and unique key constraints be part of the script that creates the table rather than be managed as a separate object in MDCMS.

However, it is recommended to key referential constraints in a separate script and managed with an attribute of type *SQLCST, so that a lock doesn’t have to be obtained on the parent table when creating the child table. Instead MDCMS will wait until the downtime period to create or automatically reapply the referential constraint.

If locking in the target environment isn't an issue, then you can include the referential constrain in the create table script and MDCMS will automatically disable the constraint before copying data to the new version of a child table.

If you wish to merely drop a constraint without otherwise changing the table, you can request the constraint with request reason Delete without needing a script or any commands defined.

*SQLFUN – Function / *SQLPRC - Procedure

Description

A function is a routine that can be invoked from within other SQL statements and returns a value or a table.

A procedure (sometimes called a stored procedure) is a routine that can be called to perform operations that can include both host language statements and SQL statements.

Sytem Type

*PGM or *SRVPGM, either pointing to an external, separately coded program with its own object request or an automatically created CLE routine containing the SQL body.

System Catalog Table

QSYS2/ SYSFUNCS for functions and QSYS2/SYSPROCS for procedures

Special Handling in MDCMS

The specific name of the routine must be name used on the object request to uniquely identify it.

High Availability Ramifications of External Routines

When external stored procedures and functions are being replicated to a backup system, the system catalog might be out of sync with the external program or service program object.

This does not apply to SQL-only stored procedures or SQL-only functions. The catalog is usually updated when the program or service program is restored to the backup system by the high availability replication software.

If this is a new stored procedure or function and the program or service program object at the time of the restore contains the catalog information within the object, the catalog is updated with the stored procedure or function information. The object might not have the catalog information within the object if the stored procedure or function was created before the external program or service program was created.

If the catalog entry already exists for the stored procedure or function during the restore, the catalog entry is not updated. This is a problem in some cases such as if a parameter were added to the stored procedure or function. The way to correct this situation is once switched to the backup system and the backup system is now the primary system, from within MDCMS, do a recompile of the stored procedure and function attribute scripts for those scripts that were created or changed since the last switch to the backup system.

Check with your high availability vendor on their support of external stored procedures and functions since it could change over time.

*SQLIDX – Index

Description

An index is a set of pointers to rows of a base table. Each index is based on the values of data in one or more table columns. An index is an object that is separate from the data in the table. When an index is created, the database manager builds this structure and maintains it automatically.

Sytem Type

*FILE

System Catalog Table

QSYS2/SYSINDEXES

Special Handling in MDCMS

None

*SQLMQT – Materialized Query Table

Description

A materialized query table is a base table created with the CREATE TABLE statement and used to contain data that is derived (materialized) from a select-statement.

Sytem Type

*FILE

System Catalog Table

QSYS2/SYSMQTSTAT

Special Handling in MDCMS

The SQL script to create the materialized query table should be using the CREATE TABLE AS (sub-select) clause.

Then, if the data in the table should be refreshed after it has been installed to each target library (recommended), define a post-installation (type=3) command for the attribute or specific object with the following command string:

RUNSQL SQL('REFRESH TABLE ##OBJLIB##/##OBJNAM##') COMMIT(*NONE)

*SQLSCR – Data Manipulation Script

Description

This attribute is to be used to execute scripts to update data within a target library. If the script attribute is applied to an object replication template, it will be executed for each target library.

Sytem Type

none

System Catalog Table

none

Special Handling in MDCMS

The RUNSQLSTM command to execute the script should be defined as a post-installation command (Type=3).

*SQLSEQ – Sequence

Description

A sequence is a stored object that simply generates a sequence of numbers in a monotonically ascending (or descending) order. Sequences provide a way to have the database manager automatically generate unique integer and decimal primary keys, and to coordinate keys across multiple rows and tables.

Sytem Type

*DTAARA

System Catalog Table

QSYS2/SYSSEQUENCES

Special Handling in MDCMS

None

*SQLTAB – Table

Description

Tables are logical structures maintained by the database manager consisting of columns and rows.

Sytem Type

*FILE

System Catalog Table

QSYS2/SYSTABLES where TABLE_TYPE = ‘T’

Special Handling in MDCMS

If the application and/or object request for a table permits it, MDCMS will automatically reapply journaling, constraints and triggers to a table when it replaces an existing table.

By default, MDCMS automatically maps the data from the prior version of the table to the new version of the table, unless the Data Origin parameter on the object request has one of these values:

- *NONE – the new version of the table will not be automatically populated with data
- *MIGRATE – data will be migrated with the changed table, replacing any data that previously existed in the table for the target library.
- A table name – the data from a table of the defined name will be mapped into the new version of the table. In this case, the name of a specific member can be specified in the Data Member parameter to copy from that member of a multi-member origin file.

If a modification is made to a partitioned table, MDCMS will automatically reapply the existing partitioning rules and partitions to the new version of the table at deployment time.

*SQLTRG – Trigger

Description

A trigger defines a set of actions that are executed automatically whenever a delete, insert, or update operation occurs on a specified table.

Sytem Type

none

System Catalog Table

QSYS2/ SYSTRIGGERS

Special Handling in MDCMS

Triggers should be added as a separate object request with its own CREATE TRIGGER script set as a Post-Install command for the *SQLTRG attribute. Any triggers defined in the same script as the CREATE TABLE are removed by MDCMS due to potential library referential issues and to avoid executing programs when migrating data.

Triggers should be removed with a separate delete request and MDCMS will drop it when the RFP is run.

If nothing changes with the triggers for a table, MDCMS will automatically reapply them when the table is deployed as long as the application or table are set to auto-reapply.

If the trigger creates a CLE program, the object authority template applied to the *SQLTRG attribute will be applied to the program.

*SQLUDT – User Defined Type

Description

A user-defined type is a data type that is defined to the database using a CREATE statement.

Sytem Type

*SQLUDT

System Catalog Table

QSYS2/SYSUDTAUTH

Special Handling in MDCMS

Data types can be added without issue within MDCMS. However, in order to replace or drop a type, there can't be any dependencies existing over it, or those dependencies will be deleted. For this reason, if requesting to delete a type, MDCMS will run the DROP statement with clause RESTRICT after all other SQL entities on the RFP have been processed.

Only request to modify a type if there aren't any dependencies over it.

*SQLVAR – User Defined Global Variable

Description

A user-defined global variable enables you to share relational data between SQL statements without the need for application logic to support this data transfer.

Sytem Type

*SRVPGM

System Catalog Table

QSYS2/ SYSVARS

Special Handling in MDCMS

Variables can be added without issue within MDCMS.

If modifying a variable, the SQL script must use the clause CREATE OR REPLACE instead of just CREATE, because MDCMS won't be able to drop the prior version of the variable if dependencies exist over it.

If requesting to delete a variable, MDCMS will run the DROP statement with clause RESTRICT after all other SQL entities on the RFP have been processed.

*SQLVW – View

Description

A view provides an alternative way of looking at the data in one or more tables.

Sytem Type

*FILE

System Catalog Table

QSYS2/SYSVIEWS

Special Handling in MDCMS

Since views can reference other views that in turn reference other views, MDCMS will automatically sort the compile sequence for views as they are assigned to an RFP. The sequence can then be manually overridden, if necessary, by editing the object request.

Manage Temporal and History Tables

What are Temporal and History Tables

A temporal table is an SQL Table that contains additional columns to track the period of time that the current values for a row are in existence.

A history table is defined with identical columns to the temporal table and provides a historical view of each row's values.

An ALTER TABLE statement is used to connect the two tables in a versioned relationship.

Once versioning is defined for the system-period temporal table, updates and deletes to it cause the version of the row prior to the change to be inserted as a row in the history table. The special row begin and row end timestamp columns are set by the system to indicate the time span when the data for the historical row was the active data.

You can write a query that will automatically return data from both the system-period temporal table and the history table.

For example, to see what the DEPARTMENT table looked like six months ago, issue the following query:

SELECT * FROM DEPARTMENT FOR SYSTEM_TIME AS OF CURRENT TIMESTAMP - 6 MONTHS

Prerequisite to Use Temporal and History Tables

The IBM i Operating System version must be at V7R3M0 or higher to take advantage of these capabilities.

Requesting a Temporal Table in MDCMS

A temporal table is requested using a *SQLTAB attribute. The object name to request should be identical to the SQL long name for the table, just as with any other *SQLTAB request.

The source for the temporal table is an SQL CREATE TABLE script that includes the additional columns.

Example source:

```
CREATE TABLE DEPARTMENT
      (DEPTNO    CHAR(3)       NOT NULL,
       DEPTNAME  VARCHAR(36)   NOT NULL,
       MGRNO     CHAR(6),
       ADMRDEPT  CHAR(3)       NOT NULL,
       LOCATION  CHAR(16),
       SOMEINT   INTEGER,
       START_TS  TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
       END_TS    TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
       TS_ID     TIMESTAMP(12) GENERATED ALWAYS
                               AS TRANSACTION START ID,
       PERIOD SYSTEM TIME (START TS, END TS),
       PRIMARY KEY (DEPTNO));

LABEL ON TABLE DEPARTMENT IS 'Departments';
```

The standard compile-time RUNSQLSTM command is sufficient for creating the table. No other commands are necessary for the temporal table.

Requesting a History Table in MDCMS

A history table is requested using a *SQLTAB attribute. The object name to request should be identical to the SQL long name for the table, just as with any other *SQLTAB request.

The source for the history table is an SQL CREATE TABLE script that includes the LIKE clause that refers to the temporal table.

Example source:

```
CREATE TABLE DEPARTMENT_HIST FOR SYSTEM NAME DEPTHIST
             LIKE DEPARTMENT;

LABEL ON TABLE DEPARTMENT_HIST IS 'Department History';
```

The standard compile-time RUNSQLSTM command is sufficient for creating the table.

The ALTER TABLE command to join the history table to the temporal should be defined as an object-specific Post-Installation (type 3) command for the history table request with Reuse Command set to true. Example of the

command:

```
RUNSQL SQL('ALTER TABLE ++OBJLIB++/DEPARTMENT
ADD VERSIONING USE HISTORY TABLE ++OBJLIB++/DEPARTMENT_HIST')
COMMIT(*NONE)
```

If standardized naming of the history table is used in relation to the temporal table, then another option is to have a separate attribute for temporal tables and to add the ALTER TABLE post-installation command to that attribute.

When checking out a temporal table, use the new attribute, but continue using the standard *SQLTAB attribute for the history table.

If the History Table is listed in MDCMS prior to Temporal Table, the Object Request record should be edited and the sort sequence set to a higher value than the value defined for the temporal table. This ensures that the temporal table is created first so that the LIKE clause for the history table picks up the correct definition.

MDTransform Limitations for Temporal and History Tables

If MDTransform is licensed for use in your organization, temporal tables and any other SQL table that contains the GENERATED ALWAYS clause can be handled by MDTransform if the following prerequisites are met:

1. A minimum OS version and PTF level must be applied on the partition. See table:

Db2 for i Enhancement	IBM i 7.5	IBM i 7.4	IBM i 7.3	IBM i 7.2
--- Enhancements from 2022 ---				
Geospatial Analytics	SF99950 Level 3	SF99704 Level 23	Not Supported	Not Supported
REMOTE TABLE	SF99950 Level 3	SF99704 Level 23	Not Supported	Not Supported
REPLICATION_OVERRIDE global variable for system generated values	SF99950 Level 3	SF99704 Level 23	Not Supported	Not Supported

2. You are using MDCMS version 8.5 or newer
3. QIBM_DB_GENCOL_OVERRIDE Function Usage is allowed either by default or for the user defined on the job description for the target promotion level.
   You can check this with command:
   WRKFCNUSG FCNID(QIBM_DB_GENCOL_OVERRIDE)

If any of the prerequisites are not met, then MDTransform must be disabled for the specific object request for a Temporal Table (but not History Table), so that MDCMS will use CPYF instead of SQL INSERT to map existing data to the new version of the table. This is because of the GENERATED ALWAYS clause on the additional columns, which would cause the existing values to be overwritten by the INSERT statement. If you don't mind having new values generated for the temporal columns in the temporal table, you can enable MDTransform for the Data Transformation screen and the *GEN special value will be used for those columns.

When using MDTransform to copy data from one environment to the other, the history table will automatically be ignored since inserts aren't allowed directly into it. For the temporal table, *GEN will be used for the GENERATED ALWAYS columns.

View Relationship between the Temporal Table and History Table in MDXREF

MDXREF automatically captures the relationship between the 2 tables in a library. If the Objects using File option is performed on the temporal table, the history table will be included in the list with Usage = HST.

Surrogate DDS to DDL Modernization in MDCMS

What is the Surrogate Method to Modernize a Database?

If DDS physical and logical files should be converted to DDL Tables, Indexes and Views in order to take advantage of SQL capabilities, the surrogate method is one option to be used to minimize impact on program code.

In the surrogate method, an SQL table with a new name is created that contains all of the existing columns from the original physical file plus any new columns. A DDS logical file is created that has the same name and record format as the old physical file and the logical file is created over the new SQL table. This way, all programs accessing that file name and format don't need to be modified or recompiled. Additionally, any existing DDS logical files will remain and also point to the new SQL table. Additional SQL indexes and views, with new names can also be added.

Steps to take to Automate the Deployment of a DB Modernization

1. Check out the existing DDS Physical file with a logical file MDCMS attribute
2. Edit the contents of the DDS source to change it from a PF to a LF over the system name of the new SQL table. Leave all fields/keys in the source so that the record format stays the same.
3. Edit the request record of the logical file and set parameter Data Origin to *NONE. *NONE indicates that records won't be copied from the prior version of the file into this file. *NONE is mandatory when switching a file from PF to LF or vice-versa.
4. Check out a new SQL Table (*SQLTAB) with the SQL name of the new table
5. Edit the request record of the SQL table and set parameter Data Origin to the name of the original physical file. At deployment time, MDCMS will then copy the data from the old PF to the new SQL table. The old PF and the new SQL Table must be in the same library. This will also work correctly when replicating to multiple libraries.
6. If converting from a single PF with multiple members to many SQL tables, edit the request record of each SQL table and set parameter Data Member to the PF member to map from during deployment.

7. If additional data manipulation needs to occur during the deployment, it is highly recommended to use the MDTransform add-on for MDCMS, which provides robust SQL INSERT/SELECT copying with the ability to easily specify column expressions per column to be used when the existing data is copied to the new SQL table.

If the converted files contain large amounts of data, it is recommended to use MDRapid to minimize the amount of time required to deploy the changes.