**UCA Collations in Db2 12.1: Usage and Recommendations**

**By Anju Kaushik, Michael Snowbell, Ian Finlay**

**Introduction**

Db2® provides an option of locale-sensitive collation for Unicode databases. Locale-sensitive UCA based collations provide cultural correctness in sorting and comparisons, and when using these collations, strings are ordered according to the Unicode Collation Algorithm (UCA) as per the Unicode standard. To implement this collation, Db2 uses open-source code: the International Components for Unicode (ICU) (https://icu.unicode.org) implementation of the UCA algorithm. ICU uses the tailoring rules for different locales and languages supplied by Unicode Common Locale Data Repository (CLDR) <https://cldr.unicode.org/>.

Db2 has supported UCA collated databases for many releases, however in Version 12, the nature of that support has changed.  As of Version 12, databases using UCA collation will always use the latest version of ICU that Db2 supports.

This document discusses the impact of Db2 refreshing the supported ICU version, procedures for database upgrade from the previous releases, and best practices for minimizing the impact of changes to ICU versions in future releases of Db2.

**Db2 12.1 and ICU Version Updates**

Starting from Db2 Version 12, the support for ICU versions has changed. Previously, Db2 maintained multiple versions of ICU to support different collations. However, starting with Db2 12.1, only the latest version of ICU is supported.

**Understanding ICU Version Impact**

Working with the latest ICU version has many advantages. For example, each updated version of ICU may have security and defect fixes as well as updated CLDR data and algorithms describing a language.  While updating the ICU version provides these advantages, changing an ICU version can impact sorting and comparison rules for databases created with UCA (CLDR) collations.

It is important to note that collation changes do not impact numeric, datetime, boolean and binary data types.  However, these may impact order and comparison rules for character/string types. On database upgrade, SQL queries using string types for ordering and comparison may see differences in the query’s result.

The following database objects, using string types (char, varchar) may be impacted on upgrading database using UCA collations:

* **Indexes on character columns:**Indexes maintain sorting order and a collation change can impact the ordering of an index.
* **Materialized Query Tables (MQTs):**An MQT may not include a character column but still include a character expression in some other manner and thus may be impacted by a change in collation.
* **Table with DPF partitioning on a character column:** In a DPF environment, a table with a partitioning key that contains a character column may no longer be partitioned correctly when moving the table to a level of Db2 with an updated ICU collation.
* **A range partitioned table with the partitioning on a character column:**A change in collation may impact the correctness of the range partitioning.
* **Multidimensional Clustering (MDC) table:**MDC tables that are clustered along dimensions that include character columns may no longer be clustered correctly.

**Why Db2 No Longer Maintains Old ICU Versions**

Until version 11.5.9, Db2 shipped all the versions of ICU that Db2 supported, to ensure consistency of query results across different Db2 releases. The following table shows the previous UCA collations and associated ICU versions that Db2 supported in version 11.5.9:

|  |  |
| --- | --- |
| Database collation | ICU version |
| UCA400\* | 3.2 |
| UCA500R1 | 3.8 |
| CLDR181 | ICU 4.4 |
| CLDR2701 | ICU 55.1 |

 Old ICU releases do not contain updates to the UCA algorithms and CLDR data. This means that databases using older ICU versions may not represent the most up-to-date representations of languages and locales. Additionally, older ICU releases may not comply with updated security standards and government regulations.

**New Approach for ICU Collated Databases**

As of Db2 Version 12, a new approach is taken for ICU collated databases:

* **Periodic ICU Version Updates**: Db2 will periodically update the ICU version to keep it up-to-date and supported.
* **Single ICU Version Support**: A particular version of Db2 will only support one version of ICU.
* **Database Upgrade and ICU Version Update**: When a database is upgraded to a new version of Db2 that supports a new ICU version, the database will use the updated (latest) ICU version.
* **Collation keyword CLDRCOLL**: Since the ICU collation of the database does not represent a specific version of ICU, the collation keyword CLDRCOLL is introduced which represents ‘CLDR collation’.

**Procedure for Upgrading ICU Collated Databases**

After you upgrade the Db2® server and clients to version 12.1 and higher, you can upgrade the databases and database applications.

If there are existing databases that have older, no longer supported UCA-based collations, such as UCA400R1, UCA500R1, CLDR181 or CLDR2701R1 (as well as previously deprecated UCA collations such as UCA400\_NO, UCA400\_LSK, UCA400\_TH), the following guidelines are provided to upgrade the database to Db2 version 12.1 and higher versions and to refresh the collation to the latest UCA collation i.e. CLDRCOLL.

**Before Upgrading**

1. **Verify Database Readiness**: Use the [db2ckupgrade](https://www.ibm.com/docs/en/db2/12.1.0?topic=commands-db2ckupgrade-check-database-upgrade) command to verify that the database is ready for upgrade.
2. **Identify Impacted Objects**: Based on special messages added for UCA/CLDR collations, identify objects that may be impacted by the ICU version change.
3. **Act on Impacted Objects**: The decision can be made prior to upgrade to modify those objects (e.g. drop certain indexes and consider recreating them on non-character data types) and to anticipate the actions Db2 will perform on upgrade and plan for tasks that might be necessary after upgrade.

**During Upgrade**

1. **Index Handling**: Indexes - explicitly as well as implicitly created indexes (e.g. for unique columns, primary and foreign keys, indexes for Multi Dimension Clustering) - that include non-binary collated character columns are marked as invalid. Due to the nature of Db2 storage, an index being marked invalid may also mark other indexes on the same table (even if the other index does not include a character column) as requiring rebuild.

Note that invalid indexes will be rebuilt as per the database manager configuration and database configuration parameter **indexrec.**

1. **MQT Handling**: MQTs are marked for refresh on reuse. Even if an MQT does not contain a character column, there may be an indirect dependency on character collation.

**After upgrade: Action needed on partitioned tables**

Upgrade itself does not modify tables that have been partitioned on a character column using a no longer supported collation and allows immediate access to these tables.  It is the user’s responsibility to evaluate and take any corrective action.   This applies specifically to tables partitioned in a DPF environment using a partitioning key that includes (non-binary) character columns as well as tables that are range partitioned based on character columns.

In both cases of data partitioned (in DPF) or range partitioned tables that used character columns for partitioning, it is possible to use the ADMIN\_MOVE\_TABLE procedure which will recreate the table using the currently supported collation.   However, one may also evaluate whether the change in collation has impacted the partitioning and take corrective action rather than move the entire table.

**Data Partitioned tables (in DPF):**

As noted, upgrade does not attempt to move rows or identify rows that are no longer on the correct node in a DPF partitioned table.  If character columns are in use to partition the data, one can use the following steps to establish whether any rows no longer use the correct partition with the new collation:

* Create a **declared** global temporary table (DGTT) that copies the partitioning keys, and their original member number, inserting them onto the correct nodes, based on the refreshed UCA Collation Library.

declare global temporary table PARTONCHARchk as  
  (select *CH*, 1 as OrigNodeNum, 1 as rank from *MYSCHEMA.PARTONCHAR*)  
  with no data  
   on commit preserve rows  
  organize by row  
  not logged  
  distribute by hash (*CH*);

In the statement above, replace the schema and table name *MYSCHEMA.PARTONCHAR* with the schema and name of the table you need to validate. Replace both occurrences of the partitioning column, *CH* with the partitioning columns used for the table you are validating. This can be 1 or more columns, depending on your original table definition.

* Populate the session temporary table, session.PARTONCHARchk, created above, with the partitioning columns from the original table being validated. (Note that the "dense\_rank()" function is required to disassociate the keys from their original member partitioning, and allow them to be redistributed. This will send all of the partitioning keys to a single member, sort them, and apply the dense\_rank() function, before redistributing them to their proper members (based on the refreshed UCA Collation). Without the "dense\_rank()", the keys are simply copied to the session temporary table on the same member.)

insert into session.PARTONCHARchk  
  select distinct *CH*, DBPARTITIONNUM(*CH*), dense\_rank() over (order by *CH*) from *MYSCHEMA.PARTONCHAR*

As described previously, the schema and table name *MYSCHEMA.PARTONCHAR* and the partitioning column *CH* must be replaced by the schema, table name and partitioning columns of the original table being validated. All 3 occurrences of the partitioning column(s) must be replaced in the insert statement above. The DBPARTITIONNUM() only requires 1 of the partitioning columns, and will report a syntax error if multiple columns are included.

* Determine the member number the partitioning keys are on, and if any keys need to be moved due to the UCA Collation Library refresh:

select OrigNodeNum,  
       DBPARTITIONNUM(*CH*) as NewNodeNum,  
       case when OrigNodeNum = DBPARTITIONNUM(*CH*) then 'NO ACTION' else 'MOVE' end as action,  
       count(1) as PartKeysOnNewNode  
  from session.PARTONCHARchk  
group by OrigNodeNum, DBPARTITIONNUM(*CH*)  
order by OrigNodeNum, DBPARTITIONNUM(*CH*);

As described previously, the partitioning column *CH* must be one of the partitioning columns of the original table in the DBPARTITIONNUM() function.

If the above query results in no rows identified, then no further action needs to be taken for the table in question.   If there are rows impacted, then one might consider using ADMIN\_MOVE\_TABLE to effectively recreate the table with the updated partitioning.  However, in the case of only a few rows being affected, one might instead choose to selectively repartition the rows in question, abiding by your organization’s practices and procedures for updating data within the database.

**Best Practices using UCA collated databases**

Using UCA collation when creating a database provides a locale-sensitive collation which provides cultural correctness.  Specifically character data under UCA collation is sorted using rules that define the locale-based character sequence, with options for specifying case-sensitivity, accent marks, text normalization, variable top attribute controls and handling of punctuation.

UCA collation attribute details are listed in <https://www.ibm.com/docs/en/db2/12.1.0?topic=collation-unicode-algorithm-based-collations> .

However, with the changes introduced in Db2 Version 12, a UCA collated database cannot assume a constant collation algorithm and order as ICU version updates for a previously created database will change collation algorithms and order, when Db2 changes its ICU level.

As a best practice, for compatibility, efficiency, long-term maintenance and easier upgrade to new Db2 modpacks and releases, it is recommended one avoid the use of string columns (char, varchar) in database objects affected by collation that could change when Db2 refreshes its ICU support level:

* **Indexes** – indexes on character columns use the database collation as its default collation. Changes in sorting order can cause incorrect ordering and comparisons, requiring the rebuilding of these indexes when moving to a version or modpack in which Db2 refreshes its ICU support level.
* **MQTs** – If MQTs are built based on collation sensitive (string) columns or in any way use character functions or casts explicitly or implicitly, then when Db2 refreshes its ICU support level, an MQT may require rebuilding.
* **Data-partitioning key** – Change in collation can affect data partitioning in DPF nodes. When moving to a Db2 release where the ICU level has changed, this may require rebuilding tables using ADMIN\_MOVE\_TABLE to ensure that the table is partitioned correctly, reflecting the collation rule change.
* **Range partitioning on character columns** - range partitioned tables may also have partitioning keys and ranges impacted by changes in the ICU collation if their range involves a character column.   Effectively repartitioning the table when at the new ICU level using ADMIN\_MOVE\_TABLE will ensure correct range partitions.
* **MDC tables** - use numeric types in MDC dimensions to avoid rebuilding of MDC tables when Db2 refreshes its ICU support level.   
    
  If ordering on data that includes characters is needed, then defining these columns as FOR BIT DATA, when possible, would provide consistent behavior across ICU releases.

**Registry Variables**

There are a couple of registry variables that you can use to customize the behavior of UCA collated database.

A new registry variable DB2\_SUPPRESS\_DDL\_WARNINGS\_FOR\_UCA\_COLLATION has been added (with a default value of YES) that if set to NO will return a new SQL20589W warning when such an object is created.

This override can be activated by issuing the following:

    db2set DB2\_SUPPRESS\_DDL\_WARNINGS\_FOR\_UCA\_COLLATION=NO

Note that as a warning, it will not prevent the creation of the object.

**Override upgrade invalidating indexes and MQTs**

It is expected that for many customer databases (some locales and options might have fewer collation changes than others) the change in ICU level will not impact the correctness of indexes and MQTs. As noted above, database upgrade will, by default, mark relevant indexes and all MQTs so that they are rebuilt. Db2 offers an override to this behavior, so that upgrade will not mark indexes and MQTs for rebuild. This provides an option for customers to handle the rebuild of these objects on their own, and address any concerns with the cost or time required to rebuild these objects, whether there are many objects, and/or very large indexes/MQTs. In such a case, it is up to the customer to validate their own indexes/MQTs and if necessary, recreate them after upgrade.

This override can be activated by issuing the following:

    db2set DB2\_DB\_UPGRADE=DB\_UPGRADE\_SKIP\_DOWNLEVEL\_ICU

prior to upgrade.

By following these guidelines and best practices, you can minimize the impact of ICU version changes on your ICU collated databases and ensure a smooth upgrade process.