Complex Updates on Metadatavalue Table Columns

The `metadatavalue` does not have all of the data needed to make decisions on how to update its columns when the decisions on how to update may be very complex or even user driven.

This guide provides complex queries to perform most of the conditions and logic in the database via SQL while potentially providing CSV exports with much of the data needed to help a user make decisions on what to update, independent of the database.

To achieve this, complex PostgreSQL-specific queries are provided along with more standards compliant SQL scripts (where possible). Notable advantages and disadvantages of each approach may also be provided.

This guide follows changes to `metadatavalue.text_lang` table column.

The Objective

The DSpace database does not enforce any structure on text_lang and users may input any data in the field. This yields potentially very significant inconsistencies that need to be analyzed and resolved. This is not fully achievable in SQL alone given that a user must make decisions on what to change. A single data set is desired to perform all operations against and must contain the appropriate ids, handles, and values while keeping the number of columns reasonably small. This set can then be exported into CSV for the decision making user to perform advanced analysis on.

PostgreSQL WITH

The PostgreSQL-specific WITH clause is used to help achieve the desired behavior in an efficient behavior when parsing the entire data set. The WITH clause is essentially a syntax for pre-processing SQL sub-queries in a temporary manner to help Postgresql perform better optimizations when executing. This also has a benefit of making it easier to show each part of the query in documentation. The downside is that as a pre-processor, it does not perform as well as a monolithic query with inline sub-queries when LIMIT is used. When LIMIT and ORDER BY are both used, the PostgreSQL-specific query out performs the standard SQL query.

Temporary View

For the purposes of this documentation as well as CSV Exporting below, the example query will be saved as a temporary view, called `view_item_metadata`. This behavior is not necessary but is very helpful. There are also issues with PostgreSQL CSV exporting that make exporting a temporary view more practical than directly exporting a SELECT query.

CSV Exporting

The CSV exporting can be achieved in two major ways in PostgreSQL.

1. `\copy ... delimiter ',' csv header force quote *;`
2. `\o ... COPY ... TO STDOUT DELIMITER ',' CSV HEADER FORCE QUOTE * ... \o`

The first way is simple and easy except for a major caveat of not support newline characters (making large queries impractical).

The second way, by default will try to write to the filesystem as the database user, which would require the script to be run as the postgresql user. To avoid this, an additional command of `\o` in conjunction with forcing output to STDOUT is used to redirect output to the clients machine.

SQL for Metadatavalue Text Language with Item, Collection, and Community

The SQL queries provided handles almost all cases, with the known exception to be that the query does not handle *community* to *community* recursion.

The columns desired are `uuid` and `handle` for each *community*, *collection*, and *item* as well as the *metadata* `id`, `text_value`, `text_lang`, `schema`*(schema name)*, `field`*(schema element)*, and `qualifier`*(schema qualifier)*.

PostgreSQL view_item_metadata VIEW

Postgresql-Specific: view_item_metadata

```
CREATE TEMPORARY VIEW view_item_metadata AS (
  WITH community_handle_to_collection AS (
    SELECT cc.community_id AS community_uuid, cc.collection_id AS collection_uuid, h.handle AS community_handle
      FROM community2collection cc
      INNER JOIN handle h ON h.resource_id = cc.community_id
  ),
  collection_community_handle AS (
    SELECT cmhc.community_uuid, cmhc.collection_uuid, cmhc.community_handle, h.handle AS collection_handle
      FROM community_handle_to_collection cmhc
      INNER JOIN handle h ON h.resource_id = cmhc.collection_uuid
  ),
  item_to_collection AS (
    (
      SELECT ci.item_id AS item_uuid, ci.collection_id AS collection_uuid
        FROM collection2item ci
    ) UNION (
      SELECT i.uuid AS item_uuid, i.owning_collection AS collection_uuid
        FROM item i
    )
  ),
  item_handle AS (
    SELECT ic.item_uuid, ic.collection_uuid, h.handle AS item_handle
      FROM item_to_collection ic
      LEFT JOIN handle h ON h.resource_id = ic.item_uuid
  ),
  item_collection_community_handle AS (
    SELECT cch.community_uuid, cch.collection_uuid, ih.item_uuid, cch.community_handle, cch.collection_handle,
ih.item_handle
      FROM item_handle ih
      LEFT JOIN collection_community_handle cch ON cch.collection_uuid = ih.collection_uuid
  ),
  schema_field AS (
    SELECT mfr.metadata_field_id AS field_id, msr.short_id AS field_schema, mfr.element AS field_name, mfr.
qualifier AS field_qualifier
      FROM metadatafieldregistry mfr
      INNER JOIN metadataschemaregistry msr ON msr.metadata_schema_id = mfr.metadata_schema_id
  ),
  metadata_schema AS (
    SELECT m.metadata_value_id AS metadata_id, m.dspace_object_id AS metadata_uuid, sf.field_id AS
metadata_field_id, m.text_value AS metadata_text_value, m.text_lang AS metadata_text_lang, sf.field_schema AS
metadata_schema, sf.field_name AS metadata_field, sf.field_qualifier AS metadata_qualifier
      FROM metadatavalue m
      INNER JOIN schema_field sf ON sf.field_id = m.metadata_field_id
  )
  SELECT icch.community_uuid, icch.collection_uuid, icch.item_uuid, ms.metadata_uuid, ms.metadata_id, ms.
metadata_field_id, icch.community_handle, icch.collection_handle, icch.item_handle, ms.metadata_text_value, ms.
metadata_text_lang, ms.metadata_schema, ms.metadata_field, ms.metadata_qualifier
    FROM item_collection_community_handle icch
    LEFT JOIN metadata_schema ms ON ms.metadata_uuid = icch.item_uuid
);
```

The above query can be replaced with the following for a more standards compliant and more efficient select when using LIMIT.

Standard SQL view_item_metadata VIEW

Standard SQL: view_item_metadata

```
CREATE TEMPORARY VIEW view_item_metadata AS (
  SELECT icch.community_uuid, icch.collection_uuid, icch.item_uuid, ms.metadata_uuid, ms.metadata_id, ms.
metadata_field_id, icch.community_handle, icch.collection_handle, icch.item_handle, ms.metadata_text_value, ms.
metadata_text_lang, ms.metadata_schema, ms.metadata_field, ms.metadata_qualifier
    FROM (
      SELECT cch.community_uuid, cch.collection_uuid, ih.item_uuid, cch.community_handle, cch.
collection_handle, ih.item_handle
        FROM (
          SELECT ic.item_uuid, ic.collection_uuid, h.handle AS item_handle
            FROM (
              (
                SELECT ci.item_id AS item_uuid, ci.collection_id AS collection_uuid
                  FROM collection2item ci
              ) UNION (
                SELECT i.uuid AS item_uuid, i.owning_collection AS collection_uuid
                  FROM item i
              )
            ) ic
            LEFT JOIN handle h ON h.resource_id = ic.item_uuid
        ) ih
        LEFT JOIN (
          SELECT cmhc.community_uuid, cmhc.collection_uuid, cmhc.community_handle, h.handle AS collection_handle
            FROM (
              SELECT cc.community_id AS community_uuid, cc.collection_id AS collection_uuid, h.handle AS
community_handle
                FROM community2collection cc
                INNER JOIN handle h ON h.resource_id = cc.community_id
            ) cmhc
            INNER JOIN handle h ON h.resource_id = cmhc.collection_uuid
        ) cch ON cch.collection_uuid = ih.collection_uuid
    ) icch
    LEFT JOIN (
      SELECT m.metadata_value_id AS metadata_id, m.dspace_object_id AS metadata_uuid, sf.field_id AS
metadata_field_id, m.text_value AS metadata_text_value, m.text_lang AS metadata_text_lang, sf.field_schema AS
metadata_schema, sf.field_name AS metadata_field, sf.field_qualifier AS metadata_qualifier
        FROM metadatavalue m
        INNER JOIN (
          SELECT mfr.metadata_field_id AS field_id, msr.short_id AS field_schema, mfr.element AS field_name,
mfr.qualifier AS field_qualifier
            FROM metadatafieldregistry mfr
            INNER JOIN metadataschemaregistry msr ON msr.metadata_schema_id = mfr.metadata_schema_id
        ) sf ON sf.field_id = m.metadata_field_id
    ) ms ON ms.metadata_uuid = icch.item_uuid
);
```

Example: Export Entire view_item_metadata as CSV

Export as CSV Example for Specific Handle

```
\o 'dspace-all_data.csv'
COPY (
  SELECT * FROM view_item_metadata
)
TO STDOUT DELIMITER ',' CSV HEADER FORCE QUOTE *
;
\o
```

The above example exports to a CSV file called `dspace-all_data.csv`.

Example: CSV Export for Getting Specific Owning Collection Handle

Export as CSV Example for Specific Handle

```
\o 'dspace-1234-owning_collecton-uuids.csv'
COPY (
  WITH desired_handle AS (
    SELECT resource_id FROM handle WHERE handle = '1969.1/1234'
  )
  SELECT uuid FROM item
  WHERE owning_collection IN (SELECT resource_id FROM desired_handle)
)
TO STDOUT DELIMITER ',' CSV HEADER FORCE QUOTE *
;
\o
```

The above example exports to a CSV file called `dspace-1234-owning_collecton-uuids.csv`. The table `collection2item` may also be needed and could be added using a UNION as done in `view_item_metadata`.

Example: Normalize Spanish Language Code

Replace 'Spanish with 'es' for Language

```
WITH target_subset AS (
  SELECT metadata_id
    FROM view_item_metadata
    WHERE TRIM(LOWER(metadata_text_lang)) in ('spanish')
)
UPDATE metadatavalue
  SET text_lang = 'es'
  WHERE metadata_value_id IN (SELECT * FROM target_subset)
;
```

The above example shows how to perform an SQL UPDATE using information that was retrieved from analyzing the data set while utilize the `view_item_metadata` VIEW.