Zusammenführung zweier System zu einem System zwei Mandanten

- Definition
  - 1. Voraussetzungen/Annahmen
  - 2. Vorbereitung
  - 3. Mandantenfähigkeit für BOs aktivieren und initialen Mandant setzen
  - 5. Datenexport
  - 6. Datenimport
  - 7. Inhaltliche Anpassungen

Definition

In diesem Artikel wird beschrieben wie zwei Systeme, die auf dem selben Nuclet basieren, zu einem gemeinsamen System mit zwei Mandanten zusammengeführt werden können. Dabei bleiben alle funktionalen Unterschiede erhalten und die Daten aus jedem System werden jeweils einem Mandanten zugeordnet, das heißt es werden zunächst keine Daten zwischen den Mandanten geteilt.

Um ein alleinstehendes System, das zukünftig von mehreren Mandanten genutzt werden soll mandantenfähig zu machen, kann wie hier unter Punkt 2, 3 und 7 beschrieben vorgegangen werden. Unbedingt sollte vorher der Artikel Mandant gelesen werden.

Es sollte außerdem überlegt werden ob das hier beschrieben Vorgehen für den konkreten Fall sinnvoll ist - abhängig von der Komplexität des Datenmodells, der Menge der Daten und den funktionalen Anforderungen an die Mandantenfähigkeit.

Die verwendete Datenbank ist PostgreSQL, ein ähnliches Vorgehen sollte aber auch für andere Datenbanken funktionieren

> Vor der Verwendung diese How-Tos oder der Einrichtung der mandantenfähigkeit generell sollte auf jeden Fall ein Backup der Datenbank erstellt werden, falls es während dessen zu Fehlern oder Problemen kommt.

1. Voraussetzungen/Annahmen

- beide Systeme verwenden die gleiche Nuclosversion
- beide Systeme verwenden die gleichen Nuclets und Nucletversionen
- beide Systeme verwenden das gleiche Datenbanksystem mit der gleichen Codierung (z.B. Postgres 12 mit UTF-8)
- beide Systeme verwenden denselben Namen für das DB-Schema (keine strikte Voraussetzung, aber wird für alle hier durchgeführten Schritte angenommen)

> Ggf. müssen noch Anpassungen für eine gemeinsam verwendete Nucletversion gemacht werden. Dabei können Unterschiede in der Funktionalität oft über Nucletparameter oder über separate Benutzergruppen, denen dann z.B. unterschiedliche Druckausgaben zugeordnet werden können, abgebildet werden.

2. Vorbereitung

Anlegen der Mandanten

Zuerst wird unter dem Menüpunkt *Administration Mandant* auf beiden Systemen jeweils ein Mandant (und damit die zugehörige Mandantenebene) angelegt.

Anschließend wird die UID der Mandantenebene auf beiden Systemen per DB-Statement angeglichen:

```
INSERT INTO t_md_mandator_level(
        struid, strname, intversion, blnshowname, datcreated, strcreated, datchanged, strchanged
) VALUES(
        'tmp', 'tmp', 1, false, current_date, 'me', current_date, 'me'
);
UPDATE t_md_mandator SET struid_t_md_mandator_level = 'tmp';
UPDATE t_md_mandator_level SET struid = '<struid auf anderem System>' WHERE struid <> 'tmp';
UPDATE t_md_mandator SET struid_t_md_mandator_level = '<struid auf anderem System>';
DELETE FROM t_md_mandator_level WHERE struid = 'tmp';
```

Mandantenabhängige Parameter und Ressourcen

Insofern nicht auf beiden Systemen (und von beiden zukünftigen Mandanten) stets der gleiche Wert verwendet werden soll, müssen Nucletparameter auf Mandantenabhängige Werte gesetzt werden. Dies erfolgt ebenfalls unter dem Menüpunkt *Administration Mandant* im entsprechenden Reiter. Die restlichen Nucletparameter sowie alle Systemparameter müssen auf beiden Systemen auf denselben Wert gesetzt werden.

Das gleiche Vorgehen wird für Ressourcen, wie z.B. Bilddateien für Logos u.ä. wiederholt.

Benutzeraccounts vorbereiten

Für die Zusammenführung der Systeme darf es keine identischen Benutzernamen geben. Eine einfache Lösung ist die Umbenennung der Benutzernamen auf einem der Systeme mit folgendem DB-Statement:

```
UPDATE t_md_user set struser = struser || '-<Mandant>';
```

Anschließend sollten auf beiden System alle User dem jeweiligen Mandanten zugeordnet werden (Mit Hilfe der Sammelbearbeitung möglich).

Konsistenz der Datenbanken prüfen

Mithilfe der Managementkonsole ein Rebuild aller Constraints durchführen. Eventuelle Inkonsistenzen müssen behoben werden. Wenn der Befehl ohne Fehlermeldungen durchläuft ist der Zustand konsistent.

```
-package <nuclet-package> -bo “<BO-Name>” -level 1
-initial <Mandantenname> [-uniqueMandator]
```

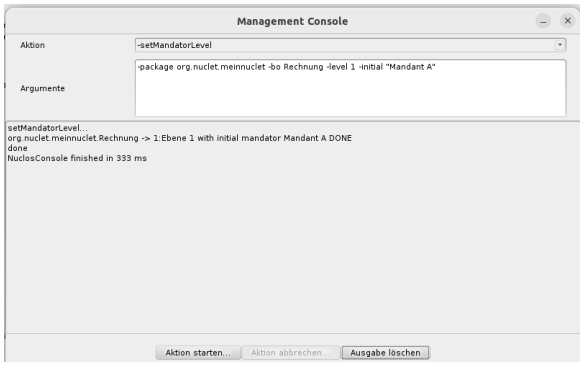
3. Mandantenfähigkeit für BOs aktivieren und initialen Mandant setzen

Im nächsten Schritt wird für die gewünschten Businessobjekte (in unserem Szenario zunächst alle) die Mandantenfähigkeit (also die Trennung der Datenbestände) aktiviert und der initiale Mandant gesetzt.

Dies ist mithilfe des folgenden Befehls auf der Managementkonsole möglich:

Wird das Flag *-uniqueMandator* verwendet, so wird der Madant Teil des unique keys der BO-Tabelle, d.h. ein Datensatz der bei der jetzigen Konfiguration ein Duplikat wäre, kann für den Datenbestand unterschiedlicher Mandanten je einmal existieren. Andernfalls bleibt der bisherige unique key mandantenübergreifend gültig.

Enthält eines der Argumente Leerzeichen, so ist es in Anführungszeichen zu setzen.



Da wir in unserem Beispiel für alle (potentiell sehr viele) BOs jeweils die gleichen Einstellungen setzen wollen, können wir dies schneller mit einem SQL-Skript erledigen. Folgendes Statement erzeugt ein solches Skript für das Nuclosschema <*schemaname*>:

migrate_entities.sql

```
set search_path to <schema_name>;

SELECT (
        'UPDATE t_md_entity SET ' || E'\n' ||
        'blnmandatorunique = true, ' || E'\n' ||
        'struid_t_md_mandator_level= ' ||
                '''' || (SELECT struid FROM t_md_mandator_level LIMIT 1) || '''' || ';' || E'\n' ||

        '-- Lege Spalte für Mandator in BO-Tabellen an'  || E'\n' ||
        (SELECT
                 STRING_AGG(
                        'ALTER TABLE ' || strdbentity || ' ADD COLUMN IF NOT EXISTS struid_nuclosmandator
character varying(128); ' ||  E'\n' ||
                         'UPDATE ' || strdbentity || ' SET struid_nuclosmandator = ' || '''' || (SELECT
struid FROM t_md_mandator LIMIT 1) || '''' || ';' || E'\n' ||
                        'ALTER TABLE ' || strdbentity || ' ALTER COLUMN struid_nuclosmandator SET NOT NULL',
                        ';' || E'\n'
                )
        FROM t_md_entity
        WHERE strvirtualentity IS NULL
        ) || ';' || E'\n'
)
```

Per shell-Befehl kann migrate_entities.sql ausgeführt, das erzeugte Skript in einer weiteren Datei zwischengespeichert und anschließend ebenfalls ausgeführt werden:

```
psql [-U "<connect with db user>"]-qAtX <DB-name> < migrate_entities.sql > do_migrate_entities.sql
psql [-U "<connect with db user>"] <DB-name> < do_migrate_entities.sql
```

Das Skript setzt für alle BOs des Systems das Mandantenlevel und den Mandator sowie die boolean-Spalte *blnmandatorunique* auf *true* (entspricht dem flag *-uniqueMandator*).

> Soll nach Ausführung des Skripts noch weitere Anpassungen an einem der Systeme vorgenommen werden müssen zunächst folgende Befehle auf der Managementkonsole ausgeführt werden, damit die Änderungen übernommen werden:
>
> 1. invalidate caches
> 2. rebuild classes
> 3. rebuild constraints and indexes

5. Datenexport

Vorbereitung

Sollten auf dem Quellsystem fehlende Dokumente (das heißt Referenzen auf nicht existierende Dokumente) existieren, sind diese zunächst zu löschen

```
DELETE FROM <schema_name>.t_md_importfile WHERE struid_documentfile = 'null';
DELETE FROM <schema_name>.t_ud_go_document WHERE struid_t_ud_documentfile = 'null';
DELETE FROM <schema_name>.t_ud_documentfile WHERE struid = 'null';
```

Export mit pg_dump

Der folgende Schritt kann möglicherweise sehr große Datenmengen produzieren. Achten sie während des Vorgangs darauf, das genug freier Speicher auf dem Zieldatenträger vorhanden ist.

Der Großteil der zu migrierenden Daten kann mit folgendem Aufruf als COPY-Statements in die Datei *my_exported_data.sql* geschrieben werden:

```
pg_dump --file "my_exported_data.sql" --host "<db host>" --port "<db port>" [--username "<connect with db
user>"] [--no-password] --verbose --format=p --blobs --data-only --no-owner --no-privileges --no-tablespaces
--no-unlogged-table-data --no-comments  --schema "<schema_name>" -t "^<schema_name>.<nuclet_prefix>*" -t
<schema_name>.t_md_mandator -t <schema_name>.t_md_mandator_accessible -t <schema_name>.t_md_user -t
<schema_name>.t_md_role_user -t <schema_name>.t_md_mandator_param_value -t <schema_name>.
t_md_mandator_role_user -t <schema_name>.t_md_passwordhistory -t <schema_name>.t_md_user_setting -t
<schema_name>.t_ud_searchfilter_user -t <schema_name>.t_ud_genericobject -t "<schema_name>.t_ud_document*" -
t "<schema_name>.t_ud_go_*" -t <schema_name>.t_ud_history -t <schema_name>.t_ud_lock  -t <schema_name>.
t_ud_entityobject_relation -t <schema_name>.t_ud_logbook -t <schema_name>.t_ud_timelimittask -t "<db name>"
```

Mit <nuclet_prefix> ist der lokale Identifizierer des Nuclets gemeint, mit dem alle Tabellennamen beginnen, die Datensätze von Businessobjekten beinhalten. Neben den Tabellen für die Businessobjekte werden Metadaten sowie Useraccounts und diesen zugeordnete Daten exportiert. Nicht migriert werden Konfigurationsdaten (d.h. im Nuclet enthaltene Objekte) sowie Systeminformationen.

Der Ausdruck *-t "^<schema_name>.<nuclet_prefix>*"* ist ggf. für jedes Nuclet das Bestandteil der Konfiguration ist zu wiederholen.

Damit der Import später funktioniert, muss das <nuclet_prefix> in der erzeugten Datei noch jeweils durch den lokalen Identifizierer des Nuclets auf dem Zielsystem ersetzt werden, z.B. mit *sed:*

```
sed -iE "s/<nuclet_prefix>/<nuclet_prefix-zielsystem>/g" my_exported_data.sql
```

Mit einem weiteren Aufruf werden die verbliebenen Daten (Tabellen sowohl Konfigurations- als auch Userdaten enthalten) exportiert. Diese werden als INSERT-Statements in die Zieldatei geschrieben. Damit ist ein Überspringen bereits existierender Zeilen beim Import möglich, dafür dauert der Im- und Export wesentlich länger als mit COPY-Statements.

```
/usr/bin/pg_dump --file "more_exported_data.sql"  --host "<db host>" --port "<db port>" [--username
"<connect with db user>"] [--no-password] --verbose --format=p --blobs --data-only --no-owner --no-
privileges --no-tablespaces --no-unlogged-table-data --no-comments --inserts --on-conflict-do-nothing --
schema "<schema_name>" -t <schema_name>.t_md_workspace  -t "<schema_name>.t_md_preference*"  "<schema_name>.
t_ud_print*" "<db name>"
```

6. Datenimport

Constraints temporär deaktivieren

Vor dem Import müssen auf der Zieldatenbank zunächst temporär alle foreign key constraints deaktiviert werden. Dafür kann analog zu den Punkten 3 und 4 mit drop_constraints.sql ein Skript erzeugt werden.

Import

Schließlich können die exportierten Daten mit folgenden Befehlen importiert werden.

```
psql [-U "<connect with db user>"] -a -v ON_ERROR_STOP=1 -f my_exported_data.sql <DB-name>
psql [-U "<connect with db user>"] -a -v ON_ERROR_STOP=1 -f more_exported_data.sql <DB-name>
```

Dokumentenverzeichnis zusammenführen

Schließlich müssen noch alle Dokumente aus dem *documents*-Verzeichnis (siehe Installation von Nuclos) des andren Systems in das *documents*-Verzeichnis des Zielsystems kopiert werden.

7. Inhaltliche Anpassungen

Einige Nucletkomponenten müssen noch händisch angepasst werden - dies betrifft die meisten Arten von SQL-Objekten, wie VLPs, Views, Funktionen und Datenquellen für Reports und Formulare. Siehe die Tabelle auf der Seite Mandant Wie diese Anpassung genau aussehen muss ist jeweils eine fachliche Frage.