SLUB
Wir führen Wissen.

Exit Strategie Rosetta

SLUB Dresden

Version 1.5.2, 2023-09-20

Inhaltsverzeichnis

Ziel . . . 1
Datenbankschema . . . 1
Installation . . . 4
Abschätzungen . . . 4
- Erste Abschätzung mit alter Version des Skriptes (2013-05-09) . . . 5
- Frühere Abschätzung mit Version vom 2017-10-17 . . . 5
- Frühere Abschätzung mit Version vom 2021-07-27 (SQLite) . . . 5
- Frühere Abschätzung mit Version vom 2021-07-28 (SQLite) . . . 5
- Frühere Abschätzung mit Version vom 2022-05-19 (SQLite) . . . 6
- Aktuelle Abschätzung mit Version vom 2022-09-11 (SQLite) . . . 6

Probleme . . . 6
- UTF-8 Bereiche in Dublincore . . . 6
- Fehlende Unterstützung mehrfacher Kopien . . . 7

Anmerkungen seitens ExLibris Rosetta . . . 7

Ziel

Um die Verfügbarkeit der langzeitarchivierten Daten auch bei einem, wie auch immer verursachten, Wegfall des Rosetta-Systems sicherzustellen, ist es notwendig rechtzeitig Vorsorge zu treffen.

Im Team wurde dazu folgende Vorgehensweise vereinbart:

1. Perl-Skript, welches über die Permanentspeicher-Verzeichnisse (zB. */permanent_storage/*) wandert
2. dabei die IE-XML-Dateien der Rosetta-AIPs parst
3. eine SQLite-Datenbank aufbaut
4. und bei Bedarf das SQL Script herausschreibt

Datenbankschema

Die zu erzeugende Datenbank soll dabei die DublinCore-Elemente der DMD-Section der AIP-Pakete (aus IE-XML-Dateien), den Namen, den tatsächlichen Speicherpfad der einzelnen Dateien und die Kopie enthalten.

Aus Performancegründen wird die Lage der IE-XML-Dateien in von den sonstigen Dateien getrennten Tabellen verwaltet.

Auf die Speicherung der Prüfsummen der Dateien wird verzichtet, da im System 3 bzw. 4 Kopien der Dateien (inklusive der IE-XML-Dateien) vorliegen und so im Falle einer Datenkorruption beim Ingest ein Mehrheitsentscheid zur Sicherstellung der Korrektheit ausreichend ist.

Als gelöscht markierte AIPs werden in eigener Tabelle gesondert behandelt.

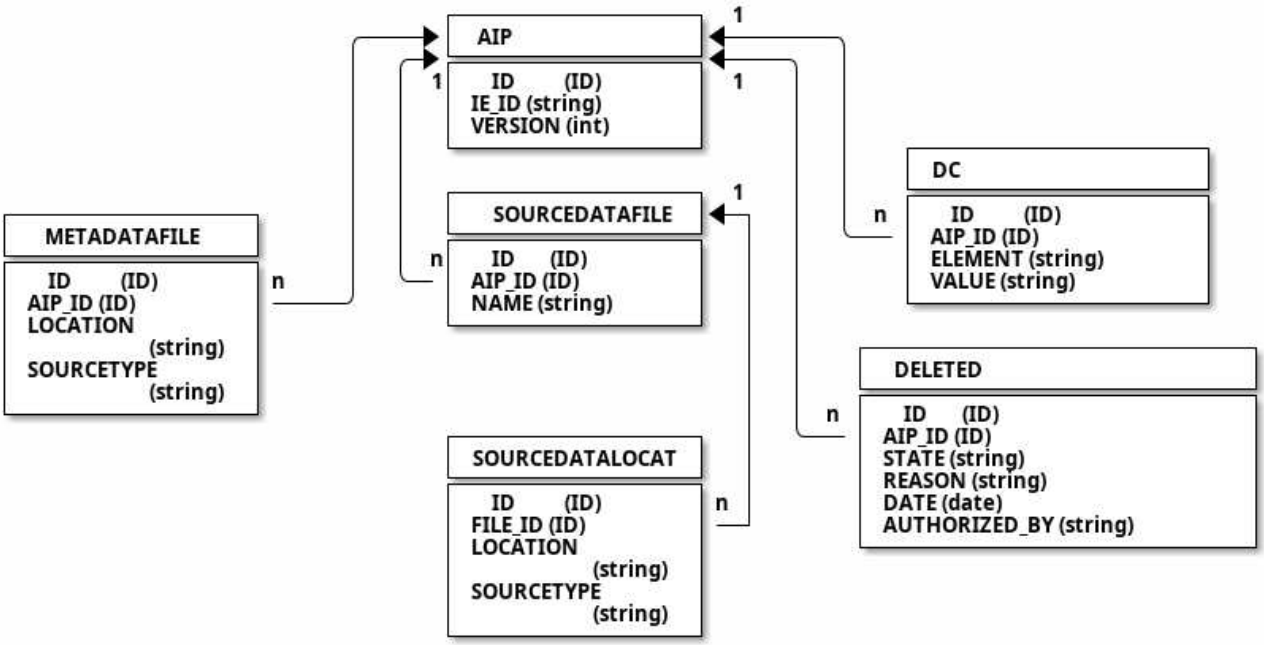

Abbildung 1. Entity Relationship Modell

Es gibt pro AIP-Eintrag in der *AIP* Tabelle eins oder mehrere *METADATAFILE*, welches die Lage der ExLibris-Rosetta-METS/MODS Datei beschreiben. Wenn mehrere Kopien abgelegt sind, gibt es mehrere Einträge.

Auch die *SOURCEDATAFILE*-Tabelle beschreibt mehrere Roh-Dateien (zB. die gescannten TIFFS der einzelnen Buchseiten), deren Kopienspeicherorte aber in der Tabelle *SOURCEDATALOCAT* hinterlegt sind.

Die wichtigsten bibliographischen Metadaten zur Suche sind in der *DC*-Tabelle hinterlegt.

Im Folgenden ist SQL für das Erzeugen der Tabellen hinterlegt:

Erzeugung der Datenbank-Tabellen

```
CREATE TABLE aip (
  id  INTEGER,
  ie_id   TEXT NOT NULL,
  version INTEGER NOT NULL,
  PRIMARY KEY(id AUTOINCREMENT)
 );
CREATE TABLE metadatafile (
  id INTEGER,
  aip_id INTEGER NOT NULL REFERENCES aip (id),
  location TEXT NOT NULL,
  sourcetype TEXT NOT NULL,
  PRIMARY KEY(id AUTOINCREMENT)
 );
CREATE TABLE dc (
  id INTEGER,
  aip_id INTEGER NOT NULL REFERENCES aip (id),
  element TEXT NOT NULL,
```

```
  value TEXT NOT NULL,
  PRIMARY KEY(id AUTOINCREMENT)
 );
CREATE TABLE sourcedatafile (
  id INTEGER,
  aip_id INTEGER NOT NULL REFERENCES aip (id),
  name TEXT NOT NULL,
  PRIMARY KEY(id AUTOINCREMENT)
 );
CREATE TABLE sourcedatalocat (
  id INTEGER,
  file_id INTEGER NOT NULL REFERENCES sourcedatafile (id),
  location TEXT NOT NULL,
  sourcetype TEXT NOT NULL,
  PRIMARY KEY(id AUTOINCREMENT)
 );
CREATE TABLE deleted (
  id INTEGER,
  aip_id INTEGER NOT NULL REFERENCES aip (id),
  state TEXT NOT NULL,
  reason TEXT NOT NULL,
  date DATE,
  authorized_by TEXT NOT NULL,
  PRIMARY KEY(id AUTOINCREMENT)
 );
CREATE UNIQUE INDEX aip_index on aip (ie_id, version);
CREATE UNIQUE INDEX sourcedata_index on sourcedatafile (aip_id, name);
```

und beispielhaft das Mapping eines AIP:

SQL Script zur Kodierung einer AIP

```
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE aip (
  id  INTEGER,
  ie_id   TEXT NOT NULL,
  version INTEGER NOT NULL,
  PRIMARY KEY(id AUTOINCREMENT)
 );
INSERT INTO aip VALUES(1,'IE96054',3);
INSERT INTO aip VALUES(2,'IE96054',2);
CREATE TABLE metadatafile (
  id INTEGER,
  aip_id INTEGER NOT NULL REFERENCES aip (id),
  location TEXT NOT NULL,
  sourcetype TEXT NOT NULL,
  PRIMARY KEY(id AUTOINCREMENT)
 );
INSERT INTO metadatafile VALUES(1,1,'V3-IE96054.xml','hdd');
INSERT INTO metadatafile VALUES(2,2,'V2-IE96054.xml','hdd');
CREATE TABLE dc (
```

```
  id INTEGER,
  aip_id INTEGER NOT NULL REFERENCES aip (id),
  element TEXT NOT NULL,
  value TEXT NOT NULL,
  PRIMARY KEY(id AUTOINCREMENT)
 );
INSERT INTO dc VALUES(1,1,'dc:title','Test StoragePlugin Rosetta 4.2.0.0');
INSERT INTO dc VALUES(2,1,'dc:creator','SLUB_Goobi');
INSERT INTO dc VALUES(3,1,'dcterms:created','06/07/2015');
INSERT INTO dc VALUES(4,1,'dc:description','Test StoragePlugin Rosetta 4.2.0.0');
INSERT INTO dc VALUES(5,2,'dc:title','Test StoragePlugin Rosetta 4.2.0.0');
INSERT INTO dc VALUES(6,2,'dc:creator','SLUB_Goobi');
INSERT INTO dc VALUES(7,2,'dcterms:created','06/07/2015');
INSERT INTO dc VALUES(8,2,'dc:description','Test StoragePlugin Rosetta 4.2.0.0');
CREATE TABLE sourcedatafile (
  id INTEGER,
  aip_id INTEGER NOT NULL REFERENCES aip (id),
  name TEXT NOT NULL,
  PRIMARY KEY(id AUTOINCREMENT)
 );
INSERT INTO sourcedatafile VALUES(1,1,'V1-FL213135.pdf');
INSERT INTO sourcedatafile VALUES(2,2,'V1-FL213135.pdf');
CREATE TABLE sourcedatalocat (
  id INTEGER,
  file_id INTEGER NOT NULL REFERENCES sourcedatafile (id),
  location TEXT NOT NULL,
  sourcetype TEXT NOT NULL,
  PRIMARY KEY(id AUTOINCREMENT)
 );
INSERT INTO sourcedatalocat VALUES(1,1,'/permanent_storage/2015/07/06/IE96054/V1-FL213135.pdf','hdd');
INSERT INTO sourcedatalocat VALUES(2,2,'/permanent_storage/2015/07/06/IE96054/V1-FL213135.pdf','hdd');
CREATE UNIQUE INDEX aip_index on aip (ie_id, version);
CREATE UNIQUE INDEX sourcedata_index on sourcedatafile (aip_id, name);
COMMIT;
```

Installation

Das Skript ist für Perl 5.28 oder höher geschrieben (älterer Perlversionen haben ua. Probleme mit UTF-8). Es verwendet unter anderem die Perl-Module *Path::Tiny*, *File::Find*, *XML::LibXML*, *DBD::SQLite*.

Dem Skript ist der Name der zu erzeugenden SQL-Datei und das Repository-Verzeichnis mitzugeben. Der Aufruf sieht so aus:

Beispiel

```
$> perl exit_strategy create_exit_database.db /permanent_storage/
```

Abschätzungen

Erste Abschätzung mit alter Version des Skriptes (2013-05-09)

Nach ersten Tests verarbeitet das Perl-Skript 277 AIPs in 112s, macht ca. 0,4s pro AIP. Es wurden dabei ca. 5200 SQL-Anweisungen erzeugt, also ca. 19 pro AIP. Das erzeugte SQL-File ist 387kB groß, pro AIP fallen also ca. 1,4kB an.

Bei anvisierten 20 Goobi Vorgängen pro Tag und Exit nach 5 Jahren würden sich ff. Werte ergeben: 35600 AIPs, Dauer ca. 4h, ca. 68000 SQL Anweisungen, 49 MB SQL-Datei.

PostgreSQL benötigte dann 5s um die 277 AIPs aus dem SQL-Skript einzulesen, hochgerechnet wäre die Datenbank dann in 11 min aufgebaut..

Frühere Abschätzung mit Version vom 2017-10-17

Für 82.000 AIPs (die im /permanent liegen) benötigt das Skript 180min, macht also 0,1s pro AIP. Es wurden dabei ca. 12 Millionen SQL Anweisungen erzeugt, also ca. 150 pro AIP. Das erzeugte SQL-File ist 1,1GB groß, pro AIP fallen also 14kB an.

Für 10.262 AIPs, die im /permanent_storage/ des Testsystems liegen, benötigt das Skript 54m25s, also 3,14s/AIP. Ausführungsumgebung ist sdvlzarossanity (virtuelle Hardware, Debian 9, 1 CPU@2.2GHz, 2GB RAM, Zugriff auf Permanentspeicher via NFSv3).

Frühere Abschätzung mit Version vom 2021-07-27 (SQLite)

Für 15.742 AIPs, die im /permanent_storage/ des Testsystems liegen, benötigt das Script 32s, also 2ms/AIP. Die erzeugte Datenbank ist 33MB groß.

Im Produktivsystem liegen zur Zeit 350.000 AIPs. Nehmen wir 14kB pro AIP an, so ist eine SQLite-Datenbank von ca. 4,4GB zu erwarten. Die Ausführungsdauer wird bei 10ms pro AIP liegen, gesamt sind also ca. 1h Prozessingzeit anzunehmen.

Frühere Abschätzung mit Version vom 2021-07-28 (SQLite)

Für 75.319 AIPs, die im /permanent_storage/ des Testsystems liegen, benötigt das Script 242s, also 3ms/AIP. Die erzeugte Datenbank ist 100MB groß. Das korrespondierende SQL ist 145MB groß.

Frühere Abschätzung mit Version vom 2022-05-19 (SQLite)

Für 252.831 AIPs des Produktivsystems benötigte das Script 23h (davon 92m15s user, 9m32s system, rest NFS wait), 3 AIPs/s (parallel zum Produktivbetrieb). Die erzeugte Datenbank ist 2,2GB groß. Das korrespondierende SQL ist 3GB groß. Pro AIP werden ca. 9kB Datenbank-Platz, bzw. 13kB SQL (ca. 130 Zeilen) benötigt.

> Eine Exit-DB wäre demnach innerhalb eines Arbeitstages prinzipiell wieder verfügbar.

Aktuelle Abschätzung mit Version vom 2022-09-11 (SQLite)

Für 771.844 AIPs des Produktivsystems benötigte das Script 38h (davon 375m8s user, 37m2s system, rest NFS wait), 5,6 AIPs/s (parallel zum Produktivbetrieb). Die erzeugte Datenbank ist 6,7 GB groß. Pro AIP werden ca. 9kB Datenbank-Platz benötigt.

Probleme

UTF-8 Bereiche in Dublincore

Es ist elementar, daß die Metadaten aus ExLibris-Rosetta sauber sind und alle Zeichen der verwendeten Dublincore-Felder als UTF-8 aus den Bereichen Basic Latin (U+0000 ⇒ U+007F), Latin-1 Supplement (U+0080 ⇒ U+00FF) und Latin-Extended-A (U+0100 ⇒ U+017F) und nicht aus anderen Bereichen stammen.

Beispielsweise wird das Zeichen ⁇ (U+FFFD) abgewiesen.

In dem Fall muß vor dem Exit eine Metadatenvalidierung innerhalb von Exlibris-Rosetta durchgeführt werden. In der Regel ist das Vorkommen von Zeichen außerhalb der oben genannten Bereiche, wie ⁇ (U+FFFD), ein Hinweis darauf, daß im Vorfeld ein Problem mit der Konvertierung zwischen UTF-8 und anderen Zeichenkodierungen vorgelegen hat.

> Relevant ist ff. Seite: Unicode Block in Java RegEx [http://docs.oracle.com/javase/7/docs/api/java/lang/Character.UnicodeBlock.html#forName%28java.lang.String%29] bzw. Unicode Pattern in Java RegEx [http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#sum]
>
> Genauer muß geprüft werden, ob ff. Unicode-Block verwendet wird: LATIN_1_SUPPLEMENT
>
> In RegEx-Notation sieht das Bspw. so aus:
>
> ```
> ^[\u0x0000-\u0x00ff]+$
> ```

Fehlende Unterstützung mehrfacher Kopien

Vor dem 2021-07-28 wird nur eine Kopie einer Datei durch das Perl-Skript unterstützt. Aktuell werden alle Kopien verarbeitet.

Anmerkungen seitens ExLibris Rosetta

Leider ist es zur Zeit so, dass seitens ExLibris noch keine offizielle Dokumentation zur Rosetta eigenen Datenablage der AIPs im Permanentspeicher vorhanden ist.

Allerdings hat ExLibris auf einen Support Incident wie folgt geantwortet:

> *Auszug aus Incident #16384-420304 SI Name: Overview / Explanation of AIP relevant files - information is requested*
>
> As you know from SI 16384-418600: "All AIPs (a.k.a. Intellectual Entities) metadata including audit (provenance) information is stored on the disk in Rosetta METS format." The actual place of the IE Rosetta METS XML files in the file system is configured in the storage rules and definitions.
>
> Home > Advanced Configuration > Repository > Storage Rules and Definitions > Storage Group List > IE Group
>
> Storage media that contains the IE METS files
>
> For example, on your staging server the NFS path to the storage 1 is */permanent_storage/ie/storage1*.
>
> The configured storages have the same structure:
>
> *<root path><storage group><storage>/<year>/<month>/<day>/<1-999 numbered subdirs with prefix>/*
>
> Example: */permanent_storage/ie/storage1/2013/03/26/file_1/*
>
> Here you find all versions of the IE Rosetta METS XML files, e.g. */permanent_storage/ie/storage1/2013/03/26/file_1/V1-IE31220.xml*
>
> The prefix *V1-* indicates that this is version #1 of the IE Rosetta METS XML file. The link to the actual file streams is in the XML in the streamref section. You have to make sure that you are using the highest (i.e. latest) version of the METS.
>
> With the information above you can develop an exit strategy which parses all IE storage directories to find the IEs and their related file streams.