Datenmodell

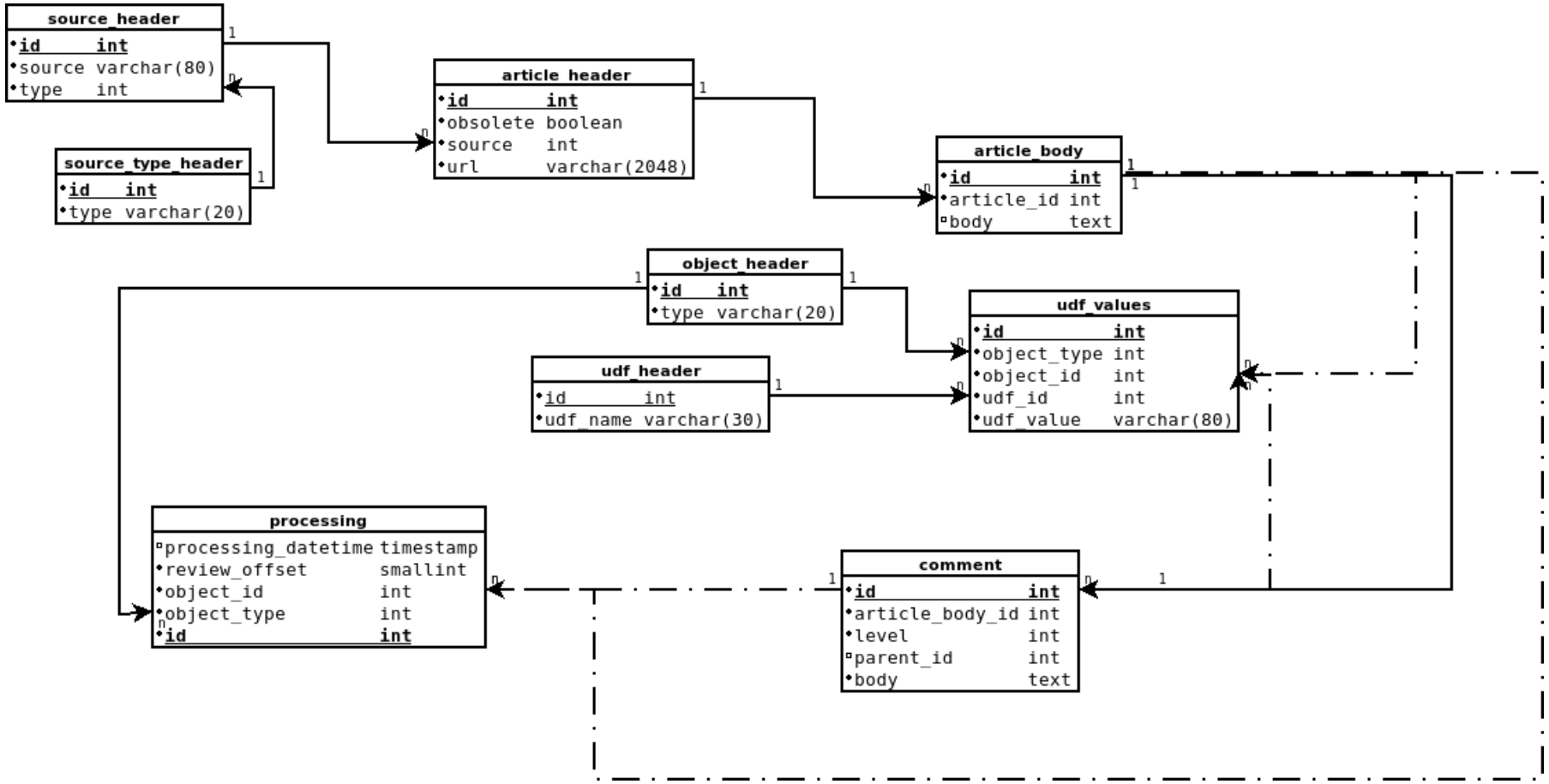
Version	Datum	erstellt/geändert von	Bemerkungen
0.1	22.11.2020	Anja	initiale Version

Inhaltsverzeichnis

open issues

Nr	Issue	Done
1	analyzer log bekommt eigene tabelle - welcher analyzer (topic detection, sentiment, ...) - object_id - object_type - timestamp_processed	
2	views - welche - sqls	

data model (overview)



tables

source_type_header

field	type	pk	nullable	unique	comment
id	int	x		x	unique identifier
type	varchar(20)				api, crawl,...

Liste der möglichen source types

```
--DROP SEQUENCE source_type_header_id_seq CASCADE;
--DROP TABLE source_type_header;

CREATE SEQUENCE source_type_header_id_seq;
CREATE TABLE source_type_header(id integer NOT NULL DEFAULT nextval('source_type_header_id_seq'), type varchar(20) NOT NULL, CONSTRAINT
pk_source_header PRIMARY KEY(id)) WITH (OIDS=FALSE);

ALTER SEQUENCE source_type_header_id_seq OWNED BY source_type_header.id;
```

source_header

field	type	pk	nullable	unique	comment
id	int	x		x	unique identifier
type	int				type of source: api, crawl
source	varchar(80)				name of source

Liste der verwendeten Quellen (Spiegel, Zeit, ...).

```
--DROP SEQUENCE source_header_id_seq CASCADE;
--DROP TABLE source_header;

CREATE SEQUENCE source_header_id_seq;
CREATE TABLE source_header(id integer NOT NULL DEFAULT nextval('source_header_id_seq'), type integer NOT NULL, source varchar(80) NOT NULL,
CONSTRAINT pk_source_header PRIMARY KEY(id), CONSTRAINT fk_source_type_header_sh FOREIGN KEY(type) REFERENCES source_type_header(id)) WITH
(OIDS=FALSE);

ALTER SEQUENCE source_header_id_seq OWNED BY source_header.id;
```

article_header

field	type	pk	nullable	unique	comment
id	int	x		x	unique identifier
source_id	int				fk source_list, source of article
url	varchar(2048)			x	url of article
obsolete	boolean				default: false; true if url is outdated (exclude from crawl / api interface)

Liste der Artikel. Es sollte jeweils nur 1 Datensatz pro Artikel abgelegt sein, soweit sich eine neuere Version ein und desselben Artikels der bereits vorhandenen zuordnen läßt. Sofern sich eine neuere Version findet, wird der Datensatz beibehalten und nur der Timestamp des "last_download" aktualisiert.

```
--DROP SEQUENCE article_header_id_seq CASCADE;
--DROP TABLE article_header;

CREATE SEQUENCE article_header_id_seq;
CREATE TABLE article_header(id integer NOT NULL DEFAULT nextval('article_header_id_seq'), source_id integer NOT NULL, url varchar(2048) NOT NULL,
obsolete boolean NOT NULL, CONSTRAINT pk_article_header PRIMARY KEY(id), CONSTRAINT fk_source_header_ah FOREIGN KEY(source_id) REFERENCES
source_header(id)) WITH (OIDS=FALSE);

ALTER SEQUENCE article_header_id_seq OWNED BY article_header.id;
```

article_body

field	type	pk	nullable	unique	comment
id	int	x		x	unique identifier
article_id	int				fk article_header
body	text				article body (max 1GB)

Sofern sich mehrere Versionen ein und desselben Artikels einander zuordnen lassen, wird der eigentlichen Inhalt des Artikels hier abgelegt und über seinen Timestamp versioniert . Die neueste Version hat den gleichen Timestamp wie der übergeordnete Artikeldatensatz und kann darüber leicht identifiziert werden.

```
--DROP SEQUENCE article_body_id_seq CASCADE;
--DROP TABLE article_body;

CREATE SEQUENCE article_body_id_seq;
CREATE TABLE article_body(id integer NOT NULL DEFAULT nextval('article_body_id_seq'), article_id integer NOT NULL, body text NOT NULL, CONSTRAINT
pk_article_body PRIMARY KEY(id), CONSTRAINT fk_article_header_ab FOREIGN KEY(article_id) REFERENCES article_header(id) ON DELETE CASCADE) WITH
(OIDS=FALSE);

ALTER SEQUENCE article_body_id_seq OWNED BY article_body.id;
```

hier wollen wir ein CASCADE; wenn der zugehörige Artikel im Header gelöscht wird

object_header

field	type	pk	nullable	unique	comment
id	int	x		x	unique identifier
type	varchar(20)				type of object (article_body, comment,...)

Liste der object_types.

```
--DROP SEQUENCE object_header_id_seq CASCADE;
--DROP TABLE object_header;

CREATE SEQUENCE object_header_id_seq;
CREATE TABLE object_header(id integer NOT NULL DEFAULT nextval('object_header_id_seq'), type varchar(20) NOT NULL, CONSTRAINT pk_object_header
PRIMARY KEY(id)) WITH (OIDS=FALSE);

ALTER SEQUENCE object_header_id_seq OWNED BY object_header.id;
```

object_header (values)

udf_header (value)	
article_body	
comment	

```
--delete from object_header;


insert into object_header (type) values('article_body');
insert into object_header (type) values('comment');
```

udf_header

field	type	pk	nullable	unique	comment
id	int	x		x	unique identifier
udf_name	varchar(30)				name of user definable field (additional information)

Liste der frei konfigurierbaren zusätzlichen Informationsfelder.

```
--DROP SEQUENCE udf_header_id_seq CASCADE;
--DROP TABLE udf_header;

CREATE SEQUENCE udf_header_id_seq;
CREATE TABLE udf_header(id integer NOT NULL DEFAULT nextval('udf_header_id_seq'), udf_name varchar(30) NOT NULL, CONSTRAINT pk_udf_header PRIMARY
KEY(id)) WITH (OIDS=FALSE);

ALTER SEQUENCE udf_header_id_seq OWNED BY udf_header.id;
```

udf_header (values)

udf_header (value)	
headline	
label	
author	
date_created	
date_modified	
date_published	
...	

```
--delete from udf_header;


insert into udf_header (udf_name) values('headline');
insert into udf_header (udf_name) values('label');
insert into udf_header (udf_name) values('author');
insert into udf_header (udf_name) values('date_created');
insert into udf_header (udf_name) values('date_modified');
insert into udf_header (udf_name) values('date_published');
```

udf_values

field	type	pk	nullable	unique	comment
id	int	x		x	unique identifier
udf_id	int				fk udf_header
object_type	int				fk object_header, object referenced by udf (article_body, comment, ...)
object_id	int				"*fk*" article_body, fk comment – fk is not enforced by structure as 2 different tables contribute their ids
udf_value	varchar(80)				user definable field (additional information)

Inhalt der frei konfigurierbaren zusätzlichen Informationsfelder, die nach Bedarf an den article_body angehängt werden können. Da nicht alle Datenquellen die gleichen Informationen liefern, ist eine Liste mit diesen Zusatzinformationen am einfachsten zu handhaben (und leicht erweiterbar).

Hier kann ggf. noch ein Flag eingefügt werden, ob der Wert aus den Rohdaten stammt oder aus unserer eigenen Verarbeitung angereichert wurde (bspw. Label)

Sofern wir feststellen, dass es notwendig ist, einen echten FK zu etablieren zwischen comment und udf_values sowie article_body und udf_values, splitte ich die Tabelle in 2 und füge den FK hinzu mit Löschweitergabe.

```
--DROP SEQUENCE udf_values_id_seq CASCADE;
--DROP TABLE udf_values;

CREATE SEQUENCE udf_values_id_seq;
CREATE TABLE udf_values(id integer NOT NULL DEFAULT nextval('udf_values_id_seq'), udf_id integer NOT NULL, object_type integer NOT NULL, object_id
integer NOT NULL, udf_value varchar(20) NOT NULL, CONSTRAINT pk_udf_values PRIMARY KEY(id), CONSTRAINT fk_udf_header_uv FOREIGN KEY(udf_id)
REFERENCES udf_header(id),CONSTRAINT fk_object_header_uv FOREIGN KEY(object_id) REFERENCES object_header(id)) WITH (OIDS=FALSE);

ALTER SEQUENCE udf_values_id_seq OWNED BY udf_value.id;
```

comment

field	type	pk	nullable	unique	comment
id	int	x		x	unique identifier
article_body_id	int				fk article_body
parent_id	int		x		null if comment on article (level=0), id of comment if comment on comment (level > 1)
level	int				0 if comment on article, incremented +1 for each sublevel (comment on comment on comment...)
body	text				comment (1 GB max)

Flexible Baumstruktur – Kommentare werden Ebenen zugewiesen. Ebene 0 ist eine direkte Reaktion auf den Artikel selbst. Ebenen 1...n sind Reaktionen auf Kommentare der übergeordneten Ebene. Der Bezug zum vorherigen Kommentar wird über parent_id hergestellt. Aktuell ist nur geplant, direkte Reaktionen auf den Artikel selbst auszuwerten (level = 0)

```
--DROP SEQUENCE comment_id_seq CASCADE;
--DROP TABLE comment;

CREATE SEQUENCE comment_id_seq;
CREATE TABLE comment(id integer NOT NULL DEFAULT nextval('comment_id_seq'), article_body_id integer NOT NULL, parent_id integer NULL, level
integer NOT NULL DEFAULT 0, body text NOT NULL, CONSTRAINT pk_comment PRIMARY KEY(id), CONSTRAINT fk_article_body_co FOREIGN KEY(article_body_id)
REFERENCES article_body(id) ON DELETE CASCADE) WITH (OIDS=FALSE);

ALTER SEQUENCE comment_id_seq OWNED BY comment.id;
```

hier wollen wir ein CASCADE; wenn der zugehörige Artikel Body gelöscht wird

processing

field	type	pk	nullable	unique	comment
id	int	x		x	unique identifier
object_id	int				"*fk*" article_body, fk comment, fk ...
object_type	int				fk object_header (type of object_id (article_body, comment,...))
processing_datetime	timestamp		x		null if not processed yet, current_datetime processing (analyzer)
review_offset	smallint				number of hours waiting time before next access of source

Wenn wir eine echten FK für die Referenz auf das Objekt haben wollen, teile ich die Tabelle in je eine pro Objekttyp. Löschweitergabe kann im Nachgang durch Einfügen von Triggern oder einen CleanUp Job umgesetzt werden.

Log, wann der Crawler/die API zuletzt Daten für das Objekt gezogen haben. Review Offset sollte nach jeder Wiederholung mit Faktor 2 erhöht werden.

```
--DROP SEQUENCE processing_id_seq CASCADE;
--DROP TABLE processing;

CREATE SEQUENCE processing_id_seq;
CREATE TABLE processing(id integer NOT NULL DEFAULT nextval('processing_id_seq'), object_id integer NOT NULL, object_type integer NOT NULL,
processing_datetime timestamp NULL, review_offset smallint NOT NULL, CONSTRAINT pk_processing PRIMARY KEY(id),
CONSTRAINT fk_object_header_pr FOREIGN KEY(object_type) REFERENCES object_header(id)) WITH (OIDS=FALSE);

ALTER SEQUENCE processing_id_seq OWNED BY processing.id;
```

Views

Alle Verknüpfungen, die unpraktisch sind in der Handhabung lösen wir hier durch Anlage von DB Views auf, sodaß Abfragen deutlich vereinfacht sind.

Liste wird sukzessive vervollständigt und die unterliegenden SQLs eingefügt.

v_processing_last_ab

```

```

v_processing_last_c

```

```

v_...