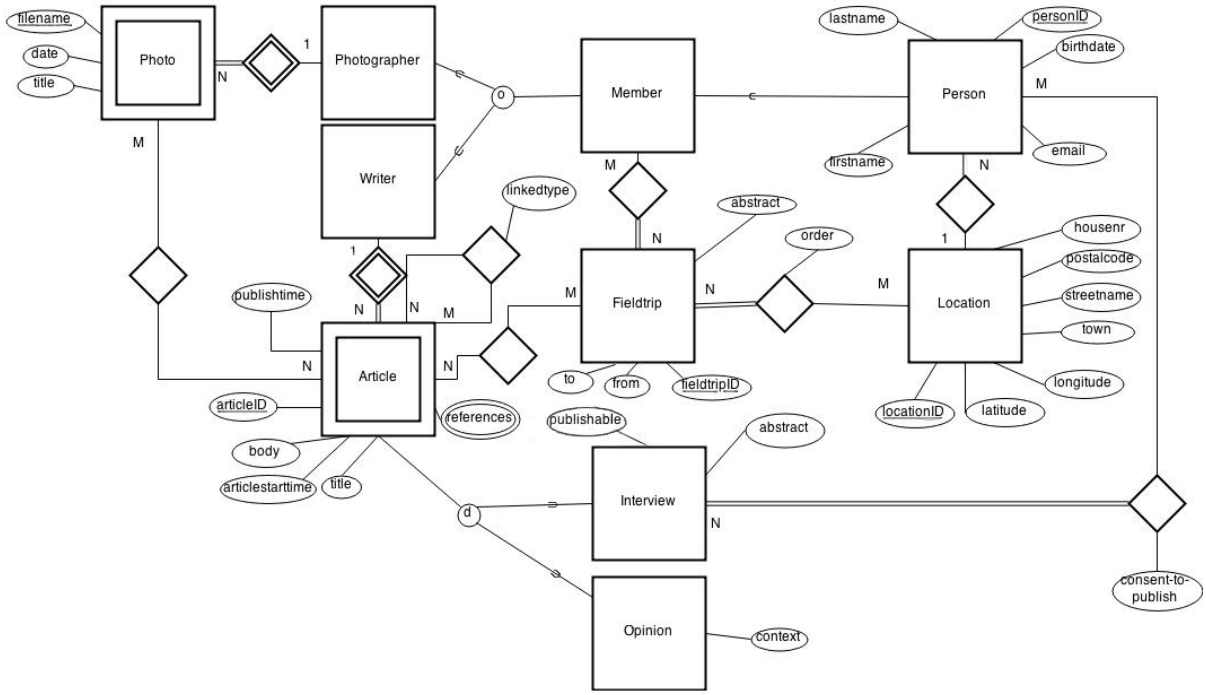
Wouter Van Steenberge, Victor Miclotte

Databases – Report Group 14

Conceptual Design

Assignment 1: (Enhanced) Entity Relationship Diagram



Assignment 2: Functional Description

In deze sectie van het verslag bespreken we de functionele beschrijving van de conceptuele databank in pseudocode.

Constraints on Person:

UNIQUE (email),

UNIQUE (firstname, lastname, birthdate),

NOT NULL (email),

NOT NULL (firstname),

NOT NULL (lastname),

CHECK (birthdate < now()).

Constraints on Photo:

DEFAULT (title) 'untitled',

DEFAULT (date) now(),

CHECK date <= now(),

CHECK date > photographer.birthdate.

Constraints on Article:

DEFAULT (title) 'untitled',

CHECK articlestarttime <= publishtime,

CHECK articlestarttime > writer.birthdate,

CHECK publishtime IS NULL WHERE publishable=false.

Constraints on Fieldtrip:

NOT NULL (from),

NOT NULL (to),

CHECK from<to.

Constraints on Location:

UNIQUE (latitude, longitude, housenr),

NOT NULL (latitude),

NOT NULL (longitude),

CHECK –90 < latitude < 90,

CHECK -180 < longitude < 180,

CHECK (NOT(housenr IS NOT NULL AND streetname IS NULL),

CHECK (NOT(streetname IS NOT NULL AND town IS NULL AND postalcode IS NULL).

Constraint on interview:

NOT NULL (abstract).

Constraint on opinion:

NOT NULL (context).

Constraint on interview_person:

DEFAULT (consent-to-publish) false,

CHECK (publishable=false AND publishtime IS NULL WHERE consent-to-publish=false) AND (publishable=true WHERE consent-to-publish=true).

Constraint on location_fieldtrip:

DEFAULT (order) 1.

Constraint on article_fieldtrip:

CHECK article.publishtimetime>=fieldtrip.from.

Constraint on member_fieldtrip (deze wordt pas toegevoegd aan de databank nadat alle data is ingelezen zodat testquery5 nog zin heeft):

CHECK fieldtrip1.to <= fieldtrip2.from OR fieldtrip1.from >= fieldtrip2.to WHERE member_fieldtrip1.personID = member_fieldtrip2.personID AND member_fieldtrip1.fieldtripID = fieldtrip1.fieldtripID AND member_fieldtrip2.fieldtripID = fieldtrip2ID.

Constraint on location_fieldtrip:

UNIQUE (fieldtripID, locationorder).

Logical Design

Assignment 3: Relational Scheme

In deze sectie bespreken we de ometting van het EER-diagram naar een relationeel schema stap voor stap.

Omzetting van reguliere entiteittypes:

1.0 **Person** (personID int, email varchar, birthdate date, firstname varchar, lastname varchar) PK: personID

2.0 **Fieldtrip** (fieldtripID int, from date, to date, abstract varchar) PK: fieldtripID

3.0 **Location** (locationID int, longitude numeric, latitude numeric, town varchar, streetname varchar, postalcode int, housenr int) PK: locationID

Omzetting van zwakke entiteittypes:

4.0 **Article** (personID int, articleID int, title varchar, body varchar, publishtime timestamp, starttime timestamp) PK: (articleID, personID) FK: personID ref: Writer

5.0 **Photo** (personID int, filename varchar, date date, title varchar) PK: (personID, filename) FK: personID ref: Photographer

Omzetting van specialisaties en generalisaties:

6.0 **Photographer** (personID int) PK: personID FK: personID ref: Member)

7.0 **Writer** (personID int) PK: personID FK: personID ref: Member

8.0 **Member** (personID int) PK: personID FK: personID ref: Person

9.0 **Opinion** (articleID int, context varchar) PK: articleID FK: articleID ref: Article

10.0 **Interview** (articleID int, abstract varchar, publishable boolean) PK: articleID FK: articleID ref: Article

Omzetting van 1-n relaties:

1.1 **Person** (personID int, locationID int, email varchar, birthdate date, firstname varchar, lastname varchar) PK: personID FK: locationID ref: Location

Omzetting van n-m relaties:

11.0 **Member_Fieldtrip** (personID int, fieldtripID int) PK: (personID, fieldtripID) FK: personID ref: Person, locationID ref: Location

12.0 **Location_Fieldtrip** (fieldtripID int, locationID int, locationorder int) PK: (fieldtripID, locationorder) FK: fieldtripID ref: Fieldtrip, locationID ref: Location

13.0 **Article_Fieldtrip** (articleID int, fieldtripID int) PK: (articleID, fieldtripID) FK: articleID ref: Article, fieldtripID ref: Fieldtrip

14.0 **Related_Articles** (articleID_1 int, articleID_2 int) PK: (articleID_1, articleID_2) FK: articleID_1 ref: Article, articleID_2 ref: Article

15.0 **Interview_Person** (articleID int, personID int) PK: (articleID, personID) FK: articleID ref: Article, personID ref: Person

16.0 **Article_Photo** (filename varchar, personID int, articleID int) PK: (filename, personID,articleID) FK: (filename, personID) ref: Photo, articleID ref: article

Omzetting van meerwaardige attributen:

17.0 **Article_References** (reference varchar, articleID int) PK: (reference, articleID) FK: articleID ref: article

Assignment 4: Behavioral Specifications

De functionele beschrijving van het conceptueel ontwerp is zó geschreven lyricsdat we deze hier rechtstreeks zouden kunnen overnemen. Dit doen we echter niet, we verwijzen gewoon naar de functionele beschrijving.

Er zijn nog enkele gedragsspecificaties die noch in de functionele beschrijving, noch in het relationeel schema terug te vinden zijn, namelijk: totale participatie.

-Bij het toevoegen van een interview, moet dit worden toegevoegd aan interview_person.

-Bij het toevoegen van een fieldtrip, moet dit worden toegevoegd aan member_fieldtrip.

-Bij het toevoegen van een fieldtrip, moet dit worden toegevoegd aan member_fieldtrip en aan location_fieldtrip.

Deze laatste drie zijn niet geïmplementeerd in de databank wegens tijdsgebrek.

PostgreSQL Implementation

Assignment 5: Make the database!

Assignment 6: SQL DDL code

```
CREATE TABLE location
(
  latitude numeric NOT NULL,
  town character varying,
  postalcode integer,
  streetname character varying,
  locationid serial NOT NULL,
  housenr character varying,
  longitude numeric NOT NULL,
  CONSTRAINT location_pkey PRIMARY KEY (locationid),
  CONSTRAINT location_latitude_longitude_housenr_key UNIQUE (latitude,
longitude, housenr),
  CONSTRAINT location_latitude_check CHECK (-90 <= latitude and latitude <=
90),
  CONSTRAINT location_longitude_check CHECK (-180 <= longitude and longitude
<= 180)
);


CREATE TABLE person
(
  email character varying NOT NULL,
  birthdate date,
  firstname character varying NOT NULL,
  lastname character varying NOT NULL,
  locationid integer,
  personid serial NOT NULL,
```

```
  CONSTRAINT person_pkey PRIMARY KEY (personid),
  CONSTRAINT person_locationid_fkey FOREIGN KEY (locationid)
      REFERENCES location (locationid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT person_email_key UNIQUE (email),
  CONSTRAINT person_firstn_lastn_birthd_key UNIQUE
(firstname,lastname,birthdate),
  CONSTRAINT person_birthdate_check CHECK (birthdate < now())
);

CREATE TABLE fieldtrip
(
  fieldtripfrom date NOT NULL,
  fieldtripto date NOT NULL,
  abstract character varying,
  fieldtripid serial NOT NULL,
  CONSTRAINT fieldtrip_pkey PRIMARY KEY (fieldtripid),
  CONSTRAINT datumcheck CHECK (fieldtripfrom < fieldtripto)
);

CREATE TABLE member
(
  personid integer NOT NULL,
  CONSTRAINT member_pkey PRIMARY KEY (personid),
  CONSTRAINT member_personid_fkey FOREIGN KEY (personid)
      REFERENCES person (personid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
);
```

```
CREATE TABLE writer
(
  personid integer NOT NULL,
  CONSTRAINT writer_pkey PRIMARY KEY (personid),
  CONSTRAINT writer_personid_fkey FOREIGN KEY (personid)
      REFERENCES member (personid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE photographer
(
  personid integer NOT NULL,
  CONSTRAINT photographer_pkey PRIMARY KEY (personid),
  CONSTRAINT photographer_personid_fkey FOREIGN KEY (personid)
      REFERENCES member (personid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE photo
(
  filename character varying NOT NULL,
  date timestamp with time zone DEFAULT now(),
  title character varying DEFAULT 'untitled'::character varying,
  personid integer NOT NULL,
  CONSTRAINT photo_pkey PRIMARY KEY (personid, filename),
  CONSTRAINT photo_personid_fkey FOREIGN KEY (personid)
      REFERENCES photographer (personid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
```

```
  CONSTRAINT photo_date_check CHECK (date <= now())
);

CREATE TABLE article
(
  title character varying DEFAULT 'untitled'::character varying,
  body character varying NOT NULL,
  starttime timestamp without time zone,
  publishtime timestamp without time zone,
  personid integer NOT NULL,
  articleid serial NOT NULL,
  CONSTRAINT article_pkey PRIMARY KEY (articleid),
  CONSTRAINT article_personid_fkey FOREIGN KEY (personid)
      REFERENCES writer (personid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT article_date_check CHECK (starttime <= publishtime)
);

CREATE TABLE opinion
(
  articleid integer NOT NULL,
  context character varying,
  CONSTRAINT opinion_articleid_fkey FOREIGN KEY (articleid)
      REFERENCES article (articleid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT opinion_pkey PRIMARY KEY (articleid)
);
```

```
CREATE TABLE interview
(
  articleid integer NOT NULL,
  abstract character varying NOT NULL,
  publishable boolean,
  CONSTRAINT interview_articleid_fkey FOREIGN KEY (articleid)
      REFERENCES article (articleid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT interview_pkey PRIMARY KEY (articleid)
);

CREATE TABLE article_fieldtrip
(
  articleid integer NOT NULL,
  fieldtripid integer NOT NULL,
  CONSTRAINT article_fieldtrip_pkey PRIMARY KEY (fieldtripid, articleid),
  CONSTRAINT article_fieldtrip_articleid_fkey FOREIGN KEY (articleid)
      REFERENCES article (articleid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT article_fieldtrip_fieldtripid_fkey FOREIGN KEY (fieldtripid)
      REFERENCES fieldtrip (fieldtripid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE article_photo
(
  filename character varying NOT NULL,
  personid integer NOT NULL,
```

```
  articleid integer NOT NULL,
  CONSTRAINT article_photo_pkey PRIMARY KEY (filename, personid, articleid),
  CONSTRAINT article_photo_articleid_fkey FOREIGN KEY (articleid)
      REFERENCES article (articleid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT article_photo_personid_fkey FOREIGN KEY (personid, filename)
      REFERENCES photo (personid, filename) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE article_references
(
  reference character varying NOT NULL,
  articleid integer NOT NULL,
  CONSTRAINT article_references_pkey PRIMARY KEY (reference, articleid),
  CONSTRAINT article_references_articleid_fkey FOREIGN KEY (articleid)
      REFERENCES article (articleid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE interview_person
(
  consenttopublish boolean DEFAULT false,
  personid integer NOT NULL,
  articleid integer NOT NULL,
  CONSTRAINT interview_person_pkey PRIMARY KEY (personid, articleid),
  CONSTRAINT interview_person_articleid_fkey FOREIGN KEY (articleid)
      REFERENCES interview (articleid) MATCH SIMPLE
```

```
    ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT interview_person_personid_fkey FOREIGN KEY (personid)
    REFERENCES person (personid) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE location_fieldtrip
(
  locationid integer NOT NULL,
  fieldtripid integer NOT NULL,
  locationorder integer DEFAULT 1,
  CONSTRAINT location_fieldtrip_pkey PRIMARY KEY (fieldtripid, locationorder),
  CONSTRAINT location_fieldtrip_fieldtripid_fkey FOREIGN KEY (fieldtripid)
    REFERENCES fieldtrip (fieldtripid) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT location_fieldtrip_locationid_fkey FOREIGN KEY (locationid)
    REFERENCES location (locationid) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE member_fieldtrip
(
  personid integer NOT NULL,
  fieldtripid integer NOT NULL,
  CONSTRAINT member_fieldtrip_pkey PRIMARY KEY (personid, fieldtripid),
  CONSTRAINT member_fieldtrip_fieldtripid_fkey FOREIGN KEY (fieldtripid)
    REFERENCES fieldtrip (fieldtripid) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION,
```

```
  CONSTRAINT member_fieldtrip_personid_fkey FOREIGN KEY (personid)
      REFERENCES member (personid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
);


CREATE TABLE related_articles
(
  articleid integer NOT NULL,
  articleid2 integer NOT NULL,
  linkedtype character varying,
  CONSTRAINT related_articles_pkey PRIMARY KEY (articleid2, articleid),
  CONSTRAINT related_articles_articleid_fkey FOREIGN KEY (articleid)
      REFERENCES article (articleid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT related_articles_articleid2_fkey FOREIGN KEY (articleid2)
      REFERENCES article (articleid) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
);
CREATE OR REPLACE FUNCTION article_fieldtrip_date_function()
  RETURNS trigger AS
$BODY$
BEGIN
if NOT EXISTS(SELECT * FROM article,fieldtrip
WHERE NEW.articleid = article.articleid AND NEW.fieldtripid = fieldtripid
AND (article.publishtime IS NULL OR article.publishtime >=
fieldtrip.fieldtripfrom)) THEN
        RAISE NOTICE 'Article % and fieldtrip % were not added because the article
was published before the fieldtrip.',NEW.articleid,NEW.fieldtripid;
        RETURN null;
```

```
END IF;
RETURN NEW;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE;


CREATE TRIGGER insert_article_fieldtrip
  BEFORE INSERT
  ON article_fieldtrip
  FOR EACH ROW
  EXECUTE PROCEDURE article_fieldtrip_date_function();




CREATE OR REPLACE FUNCTION article_publishtime_function()
  RETURNS TRIGGER AS
$BODY$
BEGIN
IF(NEW.publishtime IS NOT NULL)
THEN
        IF(NOT (SELECT publishable FROM interview WHERE articleid =
NEW.articleid))
        THEN
        RAISE EXCEPTION 'Dit artikel mag niet gepubliceerd worden!';
        END IF;
END IF;
RETURN NEW;
END;
$BODY$
```

```
  LANGUAGE plpgsql VOLATILE;


CREATE TRIGGER article_publishtime
  BEFORE UPDATE
  ON article
  FOR EACH ROW
  EXECUTE PROCEDURE article_publishtime_function();




CREATE OR REPLACE FUNCTION articlestarttime_birthdate()
  RETURNS TRIGGER AS
$BODY$
BEGIN
IF EXISTS (SELECT * FROM person WHERE NEW.personid = person.personid AND
birthdate>=NEW.starttime)
THEN RAISE EXCEPTION 'The writer wasn''t born yet when this article was
written';
END IF;
RETURN NEW;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE;


CREATE TRIGGER insert_article_datecheck
  BEFORE INSERT
  ON article
  FOR EACH ROW
  EXECUTE PROCEDURE articlestarttime_birthdate();
```

```
CREATE OR REPLACE FUNCTION interview_person_consent()
  RETURNS TRIGGER AS
$BODY$
BEGIN
IF(NEW.consenttopublish)
THEN
        IF((SELECT publishable FROM interview WHERE articleid = NEW.articleid) IS
        NULL)
        THEN
                UPDATE interview
                SET publishable = true
                WHERE NEW.articleid = articleid;
        END IF;
ELSE
        UPDATE article
        SET publishtime = null
        WHERE NEW.articleid = articleid;

        UPDATE interview
        SET publishable = false
        WHERE NEW.articleid = articleid;
END IF;
RETURN NEW;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE;
```

```
CREATE TRIGGER interview_person_consenttopublish
  BEFORE INSERT
  ON interview_person
  FOR EACH ROW
  EXECUTE PROCEDURE interview_person_consent();



CREATE OR REPLACE FUNCTION interview_person_update_consent()
  RETURNS TRIGGER AS
$BODY$
BEGIN
IF( EXISTS(SELECT * FROM interview INNER JOIN interview_person
USING(articleid)
WHERE NEW.articleid = articleid AND consenttopublish = false))
THEN
        UPDATE interview
        SET publishable = false
        WHERE articleid = NEW.articleid;
ELSE
        UPDATE interview
        SET publishable = true
        WHERE articleid = NEW.articleid;

END IF;
RETURN NEW;
END;
$BODY$
```

```
  LANGUAGE plpgsql VOLATILE;


CREATE TRIGGER interview_person_update_consent
  AFTER UPDATE
  ON interview_person
  FOR EACH ROW
  EXECUTE PROCEDURE interview_person_update_consent();




CREATE OR REPLACE FUNCTION location_fieldtrip_order_function()
  RETURNS trigger AS
$BODY$
BEGIN
IF EXISTS(SELECT * FROM (SELECT fieldtripid,max(locationorder) AS maxorder
FROM location_fieldtrip
        GROUP BY fieldtripid HAVING(fieldtripid = new.fieldtripid)) AS stuff INNER
        JOIN location_fieldtrip
        ON(maxorder = locationorder) WHERE NEW.fieldtripid =
location_fieldtrip.fieldtripid AND NEW.locationid = locationid)
THEN
        RAISE NOTICE 'De location_fieldtrip met locationid % en fieldtripid % is niet
toegevoegd. De fieldtrip bevindt zich op dat moment al op die
plaats!',NEW.locationid, NEW.fieldtripid;
        RETURN NULL;
END IF;
RETURN NEW;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE;
```

```
CREATE TRIGGER insert_location_order
  BEFORE INSERT
  ON location_fieldtrip
  FOR EACH ROW
  EXECUTE PROCEDURE location_fieldtrip_order_function();



CREATE OR REPLACE FUNCTION photodate_birthdate()
  RETURNS TRIGGER AS
$BODY$
BEGIN
IF EXISTS (SELECT * FROM person WHERE NEW.personid = person.personid AND
birthdate>=NEW.date)
THEN RAISE EXCEPTION 'The photographer wasn''t born yet when this photo was
taken';
END IF;
RETURN NEW;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE;

CREATE TRIGGER insert_photo
  BEFORE INSERT
  ON photo
  FOR EACH ROW
  EXECUTE PROCEDURE photodate_birthdate();
```

```
--Deze functie is pas toegevoegd na het inladen van de data! Anders heeft testquery5 geen zin.

CREATE OR REPLACE FUNCTION member_fieldtrip_date_function()

  RETURNS trigger AS

$BODY$

BEGIN

        IF  EXISTS(SELECT * FROM (SELECT fieldtripfrom,fieldtripto FROM fieldtrip WHERE fieldtripid = NEW.fieldtripid) AS stuff1,

        (SELECT fieldtripfrom,fieldtripto FROM member_fieldtrip

        INNER JOIN fieldtrip USING(fieldtripid)

        WHERE NEW.personid = member_fieldtrip.personid

        AND NEW.fieldtripid <> fieldtrip.fieldtripid) AS stuff2

        WHERE stuff1.fieldtripfrom <= stuff2.fieldtripto OR stuff1.fieldtripto >= stuff2.fieldtripfrom)



        THEN

                RAISE EXCEPTION 'Dubbele boeking voor member met id %',NEW.personid;

        END IF;

        RETURN NEW;

END;

$BODY$

  LANGUAGE plpgsql VOLATILE;


CREATE TRIGGER insert_member_fieldtrip

  BEFORE INSERT

  ON member_fieldtrip

  FOR EACH ROW

  EXECUTE PROCEDURE member_fieldtrip_date_function();
```

Data

Assignment 7: Insertion of the data

Om de data van het csv-bestand in te laden maakten we een tabel 'data' (zie opdracht 6) waarin we alle data staken met behulp van de import-tool van pgAdmin III. Hierna verdeelden we de data in de juiste tabellen door middel van SQL queries.

```
CREATE TABLE data
(
  articletitle character varying,
  articlebody character varying,
  articlestarttime character varying,
  articlepublishtime character varying,
  writeremail character varying,
  abstract character varying,
  context character varying,
  lastname character varying,
  firstname character varying,
  email character varying,
  birthdate character varying,
  addresstown character varying,
  addresspostalcode character varying,
  addressstreetname character varying,
  addresshousenr character varying,
  addresslatitude character varying,
  addresslongitude character varying,
  photofilename character varying,
```

```
  phototaken character varying,
  photographeremail character varying,
  articlereference character varying,
  linkedarticletitle character varying,
  linkedarticlestarttime character varying,
  linkedtype character varying,
  intervieweeemail character varying,
  interviewweegaveconsent character varying,
  fieldtriptownname character varying,
  fieldtrippostalcode character varying,
  fieldtripstreetname character varying,
  fieldtriphousenr character varying,
  fieldtriplatitude character varying,
  fieldtriplongitude character varying,
  fieldtripstopnr character varying,
  fieldtripfrom character varying,
  fieldtripto character varying
);

–-Import into data from csv-file using pgadminIII import-tool.

--Inserting all locations into location

insert into location(longitude,latitude,town,postalcode,streetname,housenr)

select distinct cast(addresslongitude as real), cast(addresslatitude as real),
addresstown, cast(addresspostalcode as integer), addressstreetname,
addresshousenr

from data

where addresslongitude is not null and addresslatitude is not null;


--Inserting all persons into Person
```

```
insert into person(email,firstname,lastname,birthdate,locationid)

select distinct email,firstname,lastname,cast(birthdate as date),locationid

from data full join location on (data.addressstreetname=location.streetname and
data.addresstown=location.town and data.addresshousenr=location.housenr)

where email is not null;


--Inserting members into Member


insert into member(personid)

select distinct personid from data inner join person on(data.email =
person.email) where data.email in

((select distinct photographeremail as email from data where photographeremail
is not null)

union

(select distinct writeremail as email from data where writeremail is not null));


--Inserting photographers into Photographer


insert into photographer(personid)

select distinct personid from data inner join person on(data.email =
person.email) where data.email in

(select distinct photographeremail as email from data where photographeremail
is not null);


--Inserting writers into Writer


insert into writer(personid)

select distinct personid from data inner join person on(data.email =
person.email) where data.email in

(select distinct writeremail as email from data where writeremail is not null);
```

```
--Inserting all photos into Photo


insert into photo(filename,date,personid)

select distinct photofilename,cast(phototaken as timestamp with time
zone),photographer.personid

from data inner join person on(photographeremail = person.email) inner join
photographer using(personid)

where (photofilename,phototaken,photographeremail) is not null;


--Inserting all articles into Article


insert into article(body,title,starttime,publishtime,personid)

select distinct articlebody,articletitle,cast(articlestarttime as timestamp without
time zone),cast(articlepublishtime as timestamp without time
zone),writer.personid

from data inner join person on(data.writeremail = person.email) inner join writer
using(personid)

where articlebody is not null;



--Inserting all interviews into Interview


insert into interview(articleid,abstract)

select distinct articleid,abstract

from data inner join article on(articlebody=body and articletitle=title)

where abstract is not null;


--Inserting all opinions into Opinion


insert into opinion(articleid,context)
```

```
select distinct articleid,context

from data inner join article on(articlebody=body and articletitle=title and
cast(articlestarttime as timestamp without time zone)=starttime)

where context is not null;


--Inserting all fieldtrips into Fieldtrip


insert into fieldtrip(fieldtripfrom ,fieldtripto)

select distinct cast(fieldtripfrom as date),cast(fieldtripto as date) from data

where fieldtripfrom is not null and fieldtripto is not null and cast(fieldtripfrom as
date)<cast(fieldtripto as date);


--Inserting data into Article_Fieldtrip


insert into article_fieldtrip(fieldtripid,articleid)

select distinct fieldtripid, articleid

from data inner join fieldtrip on(cast(data.fieldtripfrom as
date)=fieldtrip.fieldtripfrom and cast(data.fieldtripto as date)=fieldtrip.fieldtripto)
inner join article on(articletitle=title and articlebody=body);


--Inserting data into Article_Photo


insert into article_photo(articleid,filename,personid)

select distinct articleid, filename, photo.personid

from data inner join article on(articletitle=title and articlebody=body) inner join
photo on(photofilename=filename)

where articletitle is not null and photofilename is not null;



--Inserting data into Article_References
```

```
insert into article_references(articleid,reference)

select distinct articleid, articlereference

from data inner join article on(articletitle=title and articlebody=body)

where articlereference is not null;


--Inserting data into Interview_Person


insert into interview_person(personid,articleid,consenttopublish)

select distinct person.personid, articleid, cast(intervieweegaveconsent as
boolean)

from data inner join article on(articletitle=title and articlebody=body) inner join
person on(intervieweeemail=person.email)

where intervieweegaveconsent is not null;


--Inserting data into Location_Fieldtrip


insert into location_fieldtrip(locationid,fieldtripid,locationorder)

select distinct locationid, fieldtripid, cast(fieldtripstopnr as integer)

from data inner join location on (data.addressstreetname=location.streetname
and data.addresstown=location.town and
data.addresshousenr=location.housenr)

inner join fieldtrip on(cast(data.fieldtripfrom as date) = fieldtrip.fieldtripfrom and
cast(data.fieldtripto as date) = fieldtrip.fieldtripto)

where fieldtripstopnr is not null;


insert into location_fieldtrip(locationid,fieldtripid)

select distinct locationid, fieldtripid

from data inner join location on (data.addressstreetname=location.streetname
and data.addresstown=location.town and
data.addresshousenr=location.housenr)
```

```
inner join fieldtrip on(cast(data.fieldtripfrom as date) = fieldtrip.fieldtripfrom and
cast(data.fieldtripto as date) = fieldtrip.fieldtripto)

where fieldtripstopnr is null;


--Inserting data into Member_Fieldtrip


insert into member_fieldtrip(fieldtripid,personid)

select distinct fieldtripid,personid

from data inner join fieldtrip on(cast(data.fieldtripfrom as
date)=fieldtrip.fieldtripfrom and cast(data.fieldtripto as date)=fieldtrip.fieldtripto)
inner join person using(email);


--Inserting data into Related_Articles

insert into related_articles(articleid,articleid2,linkedtype)

select distinct article.articleid, article2.articleid,linkedtype

from data inner join article on(articletitle=article.title and cast(articlestarttime as
timestamp without time zone)=article.starttime)

inner join article as article2 on(linkedarticletitle=article2.title and
cast(linkedarticlestarttime as timestamp without time zone)=article2.starttime)

where linkedarticletitle is not null;

drop table data;
```