Information Systems - Lab assignment 2

Siemen Boelkens - s2788845
Hichem Bouakaz - s2525763

December 7, 2018

1 Database Normalization

Normalize the schema up to the third normal form and create it in postgresql. For each table, indicate the primary and foreign keys. Provide the scripts to generate the schema.

1.1 Unnormalized Database

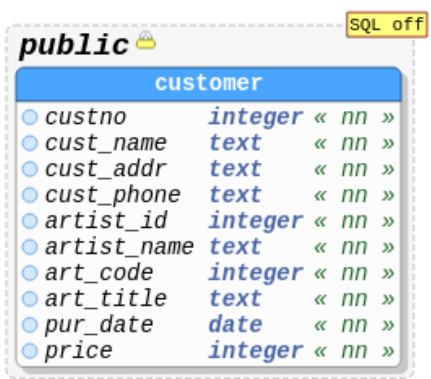

Figure 1: Unnormalized Database Schema

1.2 First Normal Database

The first normal the database should only contain atomic values at each row and column, and should have no repeating groups.
In the given example we have , $(artist\_id, artist\_name, art\_code, art\_title, pur\_date, price)$ a repeating group to change that we have introduced a primary key from the $(\text{"}custno\text{"}, \text{"}art\_code\text{"}, \text{"}pur\_date\text{"})$ column combination ,see Figure 2 .

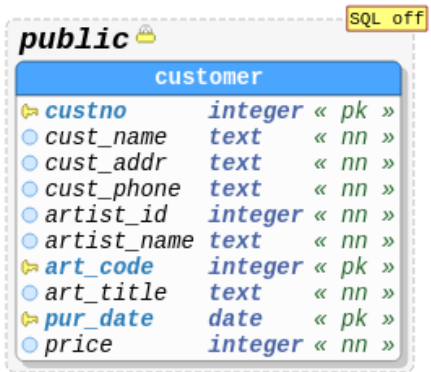

Figure 2: First normal Database Schema.

Code subsection 4.1

1.3 Second Normal Database

The second normal form requires that every nonkey attribute is fully functionally dependent on the primary key, therefore we split the table in Figure 2 into three tables where every non key is dependent on the primary key:

- **customer**: non key elements cust_name, cust_addr, cust_phone are dependent on the custno primary key.
- **purchase**: non key element price is dependent on the primary key combination (cust_no, art_code, pur_date)
- **art**: non key elements art_title, artist_id, artist_name are dependent on the primary key art_code.

See Figure 3, for the code check subsection 4.3

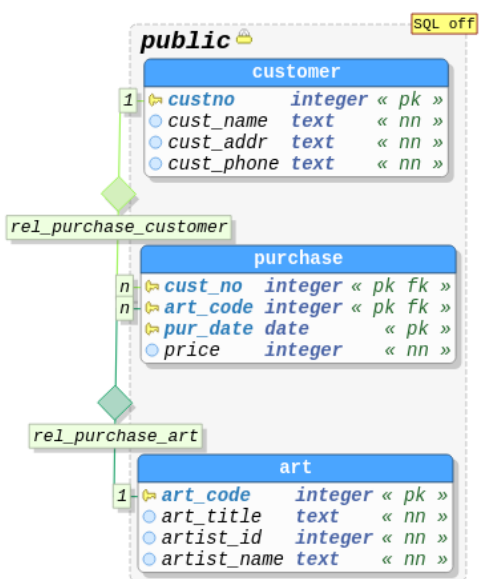

Figure 3: Second Normal Database Schema

1.4 Third Normal Database

Third Normal Form has no transitive functional dependency between nonkey attributes, to normalize the database to the third normal form we splitted the art table into art and artist since the artist_name can be determined by the artist_id, See Figure 4

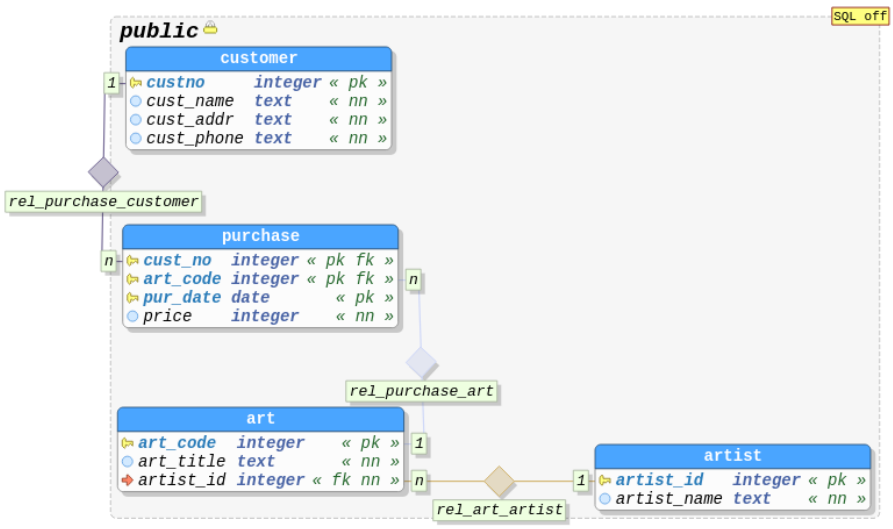

Figure 4: Third Normal Database Schema.

Related code: subsection 4.4

2 Uppercase triggers

Create triggers (and procedures/functions if necessary) in the appropriate tables to set to uppercase the customer and the artists' names with every insert or update.

```
CREATE FUNCTION public.uppercase_cust_name_on_insert ()
        RETURNS trigger
        LANGUAGE plpgsql
        VOLATILE
        CALLED ON NULL INPUT
        SECURITY INVOKER
        COST 100
        AS $$

    BEGIN
        NEW.cust_name = UPPER(NEW.cust_name);
        RETURN NEW;
    END;

$$;

CREATE TRIGGER uppercase_cust_name_on_insert
        BEFORE INSERT OR UPDATE
        ON public.customer
```

```
        FOR EACH ROW
        EXECUTE PROCEDURE public.uppercase_cust_name_on_insert();
```

```
CREATE FUNCTION public.uppercase_artist_name_on_insert ()
        RETURNS trigger
        LANGUAGE plpgsql
        VOLATILE
        CALLED ON NULL INPUT
        SECURITY INVOKER
        COST 100
        AS $$

    BEGIN
        NEW.artist_name = UPPER(NEW.artist_name);
        RETURN NEW;
    END;

$$;

CREATE TRIGGER uppercase_artist_name_on_insert_trigger
        BEFORE INSERT OR UPDATE
        ON public.artist
        FOR EACH ROW
        EXECUTE PROCEDURE public.uppercase_artist_name_on_insert();
```

3 Price trigger

Create a trigger that checks that the selling price is greater than 0.

```
CREATE FUNCTION public.validate_price_on_insert ()
        RETURNS trigger
        LANGUAGE plpgsql
        VOLATILE
        CALLED ON NULL INPUT
        SECURITY INVOKER
        COST 100
        AS $$

    BEGIN
        IF NEW.price = 0 THEN
            RAISE EXCEPTION 'The price should not be %', NEW.price;
        END IF;
        RETURN NEW;
    END;

$$;

CREATE TRIGGER validate_price_on_insert_trigger
        BEFORE INSERT OR UPDATE
        ON public.purchase
        FOR EACH ROW
```

```
EXECUTE PROCEDURE public.validate_price_on_insert();
```

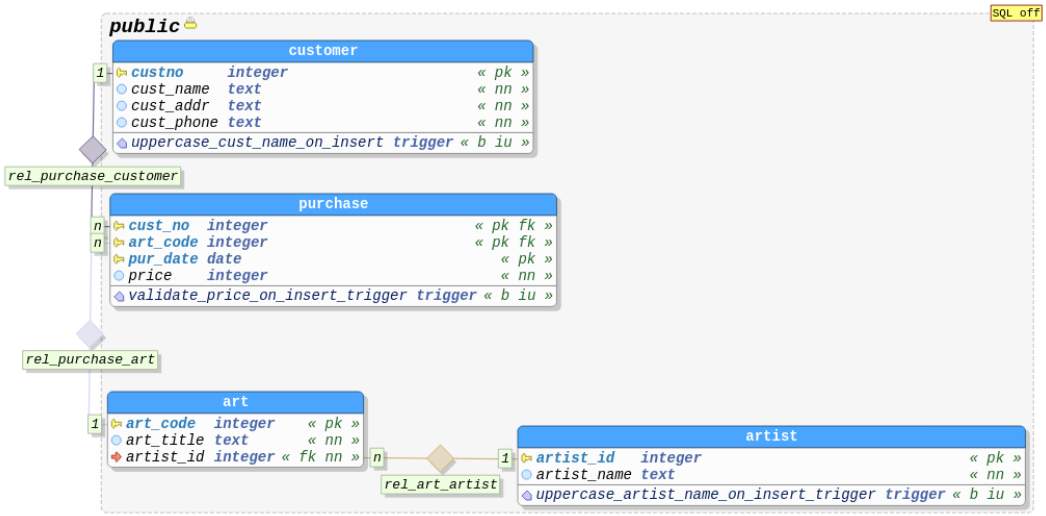

Figure 5: Third Normal Database with triggers.

4 Appendix: Code

4.1 Unnormalized Database

Unnormalized Database

```

DROP TABLE IF EXISTS "customer";
CREATE TABLE "public"."customer" (
    "custno" integer NOT NULL,
    "cust_name" text NOT NULL,
    "cust_addr" text NOT NULL,
    "cust_phone" text NOT NULL,
    "artist_id" integer NOT NULL,
    "artist_name" text NOT NULL,
    "art_code" integer NOT NULL,
    "art_title" text NOT NULL,
    "pur_date" date NOT NULL,
    "price" integer NOT NULL
) WITH (oids = false);
```

4.2 First Normal Form Database

First Normal Form Database Database

```

DROP TABLE IF EXISTS "customer";
CREATE TABLE "public"."customer" (
    "custno" integer NOT NULL,
    "cust_name" text NOT NULL,
    "cust_addr" text NOT NULL,
    "cust_phone" text NOT NULL,
    "artist_id" integer NOT NULL,
    "artist_name" text NOT NULL,
    "art_code" integer NOT NULL,
    "art_title" text NOT NULL,
    "pur_date" date NOT NULL,
    "price" integer NOT NULL,
    CONSTRAINT "customer_pkey" PRIMARY KEY ("custno", "art_code", "pur_date")
) WITH (oids = false);
```

4.3 Second Normal Form Database

Second Normal Form Database Database

```

CREATE TABLE public.art(
        art_code integer NOT NULL DEFAULT nextval('public.art_art_code_seq'::regclass),
        art_title text NOT NULL,
        artist_id integer NOT NULL,
        artist_name text NOT NULL,
        CONSTRAINT art_pk PRIMARY KEY (art_code)

);

CREATE TABLE public.customer(
        custno integer NOT NULL,
        cust_name text NOT NULL,
        cust_addr text NOT NULL,
        cust_phone text NOT NULL,
        CONSTRAINT customer_custno PRIMARY KEY (custno)

);

CREATE TABLE public.purchase(
        cust_no integer NOT NULL,
        art_code integer NOT NULL,
        pur_date date NOT NULL,
        price integer NOT NULL,
        CONSTRAINT purchase_pk PRIMARY KEY (cust_no,art_code,pur_date)

);

ALTER TABLE public.purchase ADD CONSTRAINT purchase_cust_no_fk FOREIGN KEY (cust_no)
REFERENCES public.customer (custno) MATCH FULL
ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE public.purchase ADD CONSTRAINT purchase_art_code_fk FOREIGN KEY (art_code)
REFERENCES public.art (art_code) MATCH FULL
ON DELETE NO ACTION ON UPDATE NO ACTION;
```

4.4 Third Normal Form Database

First Normal Form Database Database

```
DROP TABLE IF EXISTS "artist";
DROP SEQUENCE IF EXISTS artist_artist_id_seq;
CREATE SEQUENCE artist_artist_id_seq INCREMENT 1 MINVALUE 1 MAXVALUE 9223372036854775807 START 1 CAC

CREATE TABLE "public"."artist" (
    "artist_id" integer DEFAULT nextval('artist_artist_id_seq') NOT NULL,
    "artist_name" text NOT NULL,
    CONSTRAINT "artist_artist_id" PRIMARY KEY ("artist_id")
) WITH (oids = false);

DROP TABLE IF EXISTS "art";
DROP SEQUENCE IF EXISTS art_art_code_seq;
CREATE SEQUENCE art_art_code_seq INCREMENT 1 MINVALUE 1 MAXVALUE 9223372036854775807 START 1 CACHE 1

CREATE TABLE "public"."art" (
    "art_code" integer DEFAULT nextval('art_art_code_seq') NOT NULL,
    "art_title" text NOT NULL,
    "artist_id" integer NOT NULL,
    CONSTRAINT "art_artist_id_fkey" FOREIGN KEY (artist_id) REFERENCES artist(artist_id) NOT DEFERRA
) WITH (oids = false);



DROP TABLE IF EXISTS "customer";
CREATE TABLE "public"."customer" (
    "custno" integer NOT NULL,
    "cust_name" text NOT NULL,
    "cust_addr" text NOT NULL,
    "cust_phone" text NOT NULL,
    CONSTRAINT "customer_custno" PRIMARY KEY ("custno")
) WITH (oids = false);


DROP TABLE IF EXISTS "purchase";
CREATE TABLE "public"."purchase" (
    "cust_no" integer NOT NULL,
    "art_code" integer NOT NULL,
    "pur_date" date NOT NULL,
    "price" integer NOT NULL,
    CONSTRAINT "purchase_pkey" PRIMARY KEY ("cust_no", "art_code", "pur_date"),
    CONSTRAINT "purchase_cust_no_fkey" FOREIGN KEY (cust_no) REFERENCES customer(custno) NOT DEFERRA
) WITH (oids = false);
```

4.5 Third Normal Form Database with Triggers

First Normal Form Database Database

```

CREATE SEQUENCE public.artist_artist_id_seq
        INCREMENT BY 1
        MINVALUE 1
        MAXVALUE 9223372036854775807
        START WITH 1
        CACHE 1
        NO CYCLE
        OWNED BY NONE;

CREATE TABLE public.artist(
        artist_id integer NOT NULL DEFAULT nextval('public.artist_artist_id_seq'::regclass),
        artist_name text NOT NULL,
        CONSTRAINT artist_artist_id PRIMARY KEY (artist_id)

);
CREATE SEQUENCE public.art_art_code_seq
        INCREMENT BY 1
        MINVALUE 1
        MAXVALUE 9223372036854775807
        START WITH 1
        CACHE 1
        NO CYCLE
        OWNED BY NONE;

CREATE TABLE public.art(
        art_code integer NOT NULL DEFAULT nextval('public.art_art_code_seq'::regclass),
        art_title text NOT NULL,
        artist_id integer NOT NULL,
        CONSTRAINT art_pk PRIMARY KEY (art_code)

);
CREATE TABLE public.customer(
        custno integer NOT NULL,
        cust_name text NOT NULL,
        cust_addr text NOT NULL,
        cust_phone text NOT NULL,
        CONSTRAINT customer_custno PRIMARY KEY (custno)

);

CREATE TABLE public.purchase(
        cust_no integer NOT NULL,
        art_code integer NOT NULL,
        pur_date date NOT NULL,
        price integer NOT NULL,
        CONSTRAINT purchase_pkey PRIMARY KEY (cust_no,art_code,pur_date)

);

```

```
CREATE FUNCTION public.uppercase_cust_name_on_insert ()
       RETURNS trigger
       LANGUAGE plpgsql
       VOLATILE
       CALLED ON NULL INPUT
       SECURITY INVOKER
       COST 100
       AS $$

    BEGIN
       NEW.cust_name = UPPER(NEW.cust_name);
       RETURN NEW;
    END;

$$;

CREATE TRIGGER uppercase_cust_name_on_insert
       BEFORE INSERT OR UPDATE
       ON public.customer
       FOR EACH ROW
       EXECUTE PROCEDURE public.uppercase_cust_name_on_insert();


CREATE FUNCTION public.uppercase_artist_name_on_insert ()
       RETURNS trigger
       LANGUAGE plpgsql
       VOLATILE
       CALLED ON NULL INPUT
       SECURITY INVOKER
       COST 100
       AS $$

    BEGIN
       NEW.artist_name = UPPER(NEW.artist_name);
       RETURN NEW;
    END;

$$;

CREATE TRIGGER uppercase_artist_name_on_insert_trigger
       BEFORE INSERT OR UPDATE
       ON public.artist
       FOR EACH ROW
       EXECUTE PROCEDURE public.uppercase_artist_name_on_insert();


CREATE FUNCTION public.validate_price_on_insert ()
       RETURNS trigger
       LANGUAGE plpgsql
       VOLATILE
       CALLED ON NULL INPUT
       SECURITY INVOKER
       COST 100
       AS $$
```

```

    BEGIN
        IF NEW.price = 0 THEN
            RAISE EXCEPTION 'The price should not be %', NEW.price;
        END IF;
        RETURN NEW;
    END;

$$;

CREATE TRIGGER validate_price_on_insert_trigger
        BEFORE INSERT OR UPDATE
        ON public.purchase
        FOR EACH ROW
        EXECUTE PROCEDURE public.validate_price_on_insert();

ALTER TABLE public.art ADD CONSTRAINT art_artist_id_fkey FOREIGN KEY (artist_id)
REFERENCES public.artist (artist_id) MATCH SIMPLE
ON DELETE NO ACTION ON UPDATE NO ACTION;


ALTER TABLE public.purchase ADD CONSTRAINT purchase_cust_no_fkey FOREIGN KEY (cust_no)
REFERENCES public.customer (custno) MATCH SIMPLE
ON DELETE NO ACTION ON UPDATE NO ACTION;


ALTER TABLE public.purchase ADD CONSTRAINT purchase_art_code_fk FOREIGN KEY (art_code)
REFERENCES public.art (art_code) MATCH FULL
ON DELETE NO ACTION ON UPDATE NO ACTION;
```