

# Information System Lab 6 RDBMS

T. Back R. Bwana s3218147 s3232352

21st January, 2018

## 1 Introduction

In this assignment we carry out the process of creating and normalizing a database structure, more specifically a relational database.

The case study provided is that of an art gallery's sales database. This is first normalized up to 3 normal form and then implemented in Postgresql.

On top of this 2 triggers are created, one to ensure all names entered are stored in uppercase letters and the second to ensure that transactions are only allowed during office hours.

## 2 Normalization

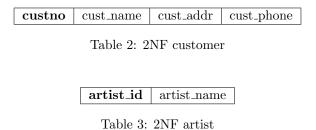
In this answer, it is assumed that the one piece of art can not be bought and sold on the same day. The primary keys or indexes are indicated in bold.

For the first normal form (1NF), all cells can only contain atomic values. This means that the in the assignment indicated brackets get separated into individual cells (atomic values). The resulting table is shown in Table 1.

 custno
 cust\_name
 cust\_addr
 cust\_phone
 artist\_id
 artist\_name
 art\_code
 art\_title
 pur\_date
 price

Table 1: 1NF

The second normal form (2NF) requires each non-key attribute to be fully functionally dependent on the primary key. Therefore, all non-keys that are identifiable by a primary key are getting moved into their own table. This results in the three additional tables Table 2, Table 3, Table 4 as well as a reduced 'main' table shown in Table 5.



art\_code | art\_title

Table 4: 2NF art

custno artist_id	art_code	pur_date	price
------------------	----------	----------	-------

Table 5: 2NF

The third normal form (3NF) requires no transitive functional dependent between non-keys. In the example this is still given with art\_code and artist\_id. So, while Table 2, Table 3 and Table 4 are kept, Table 6 gets introduced. This reduced the 'main' table to the table shown in Table 7.

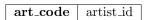


Table 6: 2NF art-artist mapping

$custno   art\_code$	pur_date	price
----------------------	----------	-------

Table 7: 3NF

# 3 Postgres

In this step, all the previous defined schema gets implemented in Postgres.

#### 3.1 Schema

To create the schema, the following code is necessary. It includes the tables, foreign keys and indexes.

```
DROP TABLE IF EXISTS "artcode";
   CREATE TABLE "public". "artcode" (
2
        "art_code" integer NOT NULL,
3
        "ard_name" character varying NOT NULL,
4
        CONSTRAINT "artcode_art_code" PRIMARY KEY ("art_code")
6
   ) WITH (oids = false);
9
   DROP TABLE IF EXISTS "artists";
10
   CREATE TABLE "public". "artists" (
        "artist_id" integer NOT NULL,
11
12
        "artist_name" character varying NOT NULL,
        CONSTRAINT "artists_artist_id" PRIMARY KEY ("artist_id")
13
14
   ) WITH (oids = false);
15
16
   DROP TABLE IF EXISTS "customers";
17
   CREATE TABLE "public". "customers" (
18
        "cust_no" integer NOT NULL,
19
20
        "cust_name" character varying NOT NULL,
        "cust_addr" character varying NOT NULL,
21
22
        "cust_phone" character varying NOT NULL,
23
        CONSTRAINT "customers_cust_no" PRIMARY KEY ("cust_no")
24
   ) WITH (oids = false);
25
26
   DROP TABLE IF EXISTS "artcode_artist_mapping";
   CREATE TABLE "public"."artcode_artist_mapping" (
28
29
        "art_code" integer NOT NULL,
        "artist_id" integer NOT NULL,
30
        CONSTRAINT "artcode_artist_mapping_art_code_fkey" FOREIGN KEY (
31
            art_code) REFERENCES artcode(art_code) NOT DEFERRABLE,
32
        CONSTRAINT "artcode_artist_mapping_artist_id_fkey" FOREIGN KEY
            (artist_id) REFERENCES artists(artist_id) NOT DEFERRABLE
33
   ) WITH (oids = false);
34
35
   CREATE INDEX "artcode_artist_mapping_art_code" ON "public"."
        artcode_artist_mapping" USING btree ("art_code");
36
    CREATE INDEX "artcode_artist_mapping_artist_id" ON "public"."
        artcode_artist_mapping" USING btree ("artist_id");
37
38
39
   DROP TABLE IF EXISTS "purchases";
   CREATE TABLE "public"."purchases"
"cust_no" integer NOT NULL,
40
41
       "art_code" integer NOT NULL,
42
        "pur_date" date NOT NULL,
43
44
        "price" money NOT NULL,
45
        CONSTRAINT "purchases_art_code_fkey" FOREIGN KEY (art_code)
            REFERENCES artcode(art_code) NOT DEFERRABLE,
46
        CONSTRAINT "purchases_cust_no_fkey" FOREIGN KEY (cust_no)
            REFERENCES customers(cust_no) NOT DEFERRABLE
```

```
47 ) WITH (oids = false);
48
49 CREATE INDEX "purchases_art_code" ON "public"."purchases" USING btree ("art_code");
50 CREATE INDEX "purchases_cust_no" ON "public"."purchases" USING btree ("cust_no");
51 CREATE INDEX "purchases_pur_date" ON "public"."purchases" USING btree ("pur_date");
```

### 3.2 Trigger - Uppercase Insert

Additionally, also triggers are introduced to ensure that on each insert or update, the name of the artist and customer is written in capital letters. For this, first a function is written and then the trigger registered.

```
CREATE OR REPLACE FUNCTION uppercase_artist_insert() RETURNS
1
       trigger AS $uppercase_artist_insert$
2
       BEGIN
3
           NEW.artist_name = UPPER(NEW.artist_name);
4
           RETURN NEW;
       END:
5
6
   $uppercase_artist_insert$ LANGUAGE plpgsql;
7
8
   CREATE OR REPLACE FUNCTION uppercase_customer_insert() RETURNS
       trigger AS $uppercase_customer_insert$
9
10
           NEW.cust_name = UPPER(NEW.cust_name);
11
           RETURN NEW:
       END;
12
   $uppercase_customer_insert$ LANGUAGE plpgsql;
13
14
   DROP TRIGGER IF EXISTS "artist_insert" ON "public"."artists";
15
   CREATE TRIGGER "artist_insert" BEFORE INSERT OR UPDATE ON "public
16
       "."artists" FOR EACH ROW EXECUTE PROCEDURE
       uppercase_artist_insert();
17
   DROP TRIGGER IF EXISTS "customer_insert" ON "public"."customers";
18
   CREATE TRIGGER "customer_insert" BEFORE INSERT OR UPDATE ON "public
19
       "."customers" FOR EACH ROW EXECUTE PROCEDURE
       uppercase_customer_insert();
```

# 3.3 Trigger - Out of office hour transactions

Finally, triggers for all tables in the database are registered that prevent any form of modification in the database. The base function office\_hours contains the necessary logic to check the office hours.

```
1 CREATE OR REPLACE FUNCTION office_hours() RETURNS trigger AS
$office_hours$
2 BEGIN
3 IF (TO_CHAR(SYSDATE,'hh:mi') NOT BETWEEN '09:00' AND
'17:00')
```

```
THEN RAISE EXCEPTION 'no modifications outside of
4
                    office hours';
           END IF:
5
6
           return NEW;
       END:
7
8
   $office_hours$ LANGUAGE plpgsql;
9
10
   DROP TRIGGER IF EXISTS "out_of_office_customers" ON "public"."
   DROP TRIGGER IF EXISTS "out_of_office_customers" ON "public"."
11
       artcode_artist_mapping";
   DROP TRIGGER IF EXISTS "out_of_office_customers" ON "public"."
12
       artists";
   DROP TRIGGER IF EXISTS "out_of_office_customers" ON "public"."
13
       customers";
   DROP TRIGGER IF EXISTS "out_of_office_customers" ON "public"."
14
       purchases";
   CREATE TRIGGER out_of_office_customers BEFORE DELETE OR INSERT OR
       UPDATE ON "public"."artcode" FOR EACH ROW EXECUTE PROCEDURE
       office_hours();
16
   CREATE TRIGGER out_of_office_customers BEFORE DELETE OR INSERT OR
       UPDATE ON "public"."artcode_artist_mapping" FOR EACH ROW
       EXECUTE PROCEDURE office_hours();
   CREATE TRIGGER out_of_office_customers BEFORE DELETE OR INSERT OR
17
       UPDATE ON "public"."artists" FOR EACH ROW EXECUTE PROCEDURE
       office_hours();
   CREATE TRIGGER out_of_office_customers BEFORE DELETE OR INSERT OR
18
       UPDATE ON "public"."customers" FOR EACH ROW EXECUTE PROCEDURE
       office_hours();
19
   CREATE TRIGGER out_of_office_customers BEFORE DELETE OR INSERT OR
       UPDATE ON "public"."purchases" FOR EACH ROW EXECUTE PROCEDURE
       office hours():
```

# A Create a postgres db

Use the following stack to deploy postgres to docker.

```
# Use postgres/example user/password credentials
1
2
    version: '3.1'
3
4
   services:
5
6
     db:
        image: postgres
7
8
        restart: always
9
        environment:
10
          POSTGRES_PASSWORD: example
11
12
      adminer:
13
        image: adminer
14
        restart: always
15
        ports:
          - 8080:8080
16
```

The commands for starting are:

```
docker swarm init
docker stack deploy -c postgres.yml postgres

# User postgres
# Pass example

# Then open localhost:8080
# Select PostGres as database type and login
```